PRIME

Initial Documentation Release

IDR 3043
PRIME COMPUTER USER'S GUIDE FOR THE DATABASE ADMINISTRATOR

P/N IDR 3062

First Printing July 1977

CONTENTS

Section	Title	Page
SECTION 1	INTRODUCTION	1-1
	ABOUT THIS MANUAL	1-1
	PRIME DBMS DOCUMENTATION	1-1
	Documentation Releases	1-3
	OTHER RELATED PRIME DOCUMENTS	1-3
	LIST OF DBMS REFERENCE DOCUMENTS	1-5
SECTION 2	PRIME'S DBMS ENVIRONMENT	2-1
	INTRODUCTION	2-1
	HARDWARE	2-1
	Speed	2-1
	Integrity and Security	2-2
	Hardware Upgrade and Expansion	2-3
	PRIMOS ARCHITECTURE	2-3
	File and Access Management	2-4
	Integrity in PRIMOS	2-4
	Security Within PRIMOS	2-4
	Buffer Pooling	2-4
	Data Structure Assistance	2-5
	Speed	2-5
	Maintainability	2-5
	DBMS ARCHITECTURE	2-5
SECTION 3	STORAGE STRUCTURE SUPPORT	3-1
	INTRODUCTION	3-1
	Database Layout	3-1
	Area Format	3-1
	Set Format & Search Keys	3-4
	Location Mode	3-6
	Ease of Expansion	3-6
	Space Utilization	3-7
SECTION 4	THE PRIME VIEW OF SOME COMMON DBMS CONCEPTS	4-1
	INTRODUCTION	4-1
	Data Structures	4-1

Access Strategies 4-1
Data Independence 4-2
Privacy 4-3
Integrity 4-3
Sharing/Concurrent Access with Most DBMS 4-3
Sharing/Concurrent Access with Prime's DBMS 4-4
Preserving the Physical Integrity of the Database 4-6

RECOVERY 4-6

The Operation of the Run-Unit 4-6
Roll Back 4-7
The Recovery Processor 4-7
Recovery Facilities 4-7

DATABASE ADMINISTRATOR SUPPORT 4-8

Privacy-Locking 4-8
Caretaker Functions 4-8
Creating the Database 4-9

SECTION 5 EXTENSIONS/RESTRICTIONS 5-1

INTRODUCTION 5-1
GLOBAL EXTENSIONS 5-1

Database Transactions (DBT'S) 5-1
INVOKE Command 5-1
Privacy Keys 5-1
ON ERROR Clause 5-1
Dynamic Reference 5-1

SPECIFIC EXTENSIONS 5-2

Item Type Extensions 5-2
Data Aggregates 5-2
FORTRAN Record Overlays 5-3

OMISSIONS/RESTRICTIONS 5-3
PENDING 5-3

SECTION 6 DBMS FILES 6-1

INTRODUCTION 6-1
SCHEMA TABLE 6-1
SUBSCHEMA TABLE 6-1
AREA FILE 6-2
SET FILE 6-2
CALC FILE 6-2
LOG FILE 6-2
BEFORE-IMAGE FILE 6-2

AFTER-IMAGE FILE 6-3

SECTION 7 DATABASE GENERATION 7-1

INTRODUCTION 7-1
SCHEMA CREATION 7-1
FILE ALLOCATION 7-1

SECTION 8 DATABASE MAINTENANCE TECHNIQUES 8-1

INTRODUCTION 8-1
MEDIA SPACE VERIFICATION 8-1
MEDIA SPACE CLEAN-UP AND EXPANSION 8-1
CONCURRENCY 8-2
RECOVERY 8-2
DATABASE BACKUP 8-3
DML COMMAND PROCESSOR CLEAN-UP 8-3
PRIVACY KEYS 8-3
EXPANSION DIALOG 8-4
OTHER ALTERATIONS 8-4

SECTION 9 DBA COMMAND PROCESSOR (DBACP) 9-1

INTRODUCTION 9-1
PROCEDURES 9-1

Concurrent Run-Units 9-1

SECURITY 9-2

Data Administrators 9-2
Schema Privacy Locks 9-2

COMMAND SYNTAX 9-2
DBACP COMMAND VERBS AND OBJECTS 9-3
COMMAND DESCRIPTION 9-5
AFTER-IMAGING 9-6

Function 9-6
Format 9-6

AREA 9-7

Function 9-7
Format 9-7

AREAS 9-9

Function 9-9
Format 9-9

BEFORE-IMAGING 9-10

Function 9-10
Format 9-10

CALC OF RECORD 9-11

Function 9-11
Format 9-11

CALCS 9-12

Function 9-12
Format 9-12

FILES 9-13

Function 9-13
Format 9-13

KEY OF LOCK 9-15

Function 9-15
Format 9-15

KEYS 9-16

Function 9-16
Format 9-16

LISTING 9-17

Function 9-17
Format 9-17

LOGGING 9-18

Function 9-18
Format 9-18

SCHEMA 9-19

Function 9-19
Format 9-19

SCHEMAS 9-21

Function 9-21
Format 9-21

SET 9-22

Function 9-22
Format 9-22

SETS 9-23

Function 9-23
Format 9-23

SUBSCHEMA 9-24

Function 9-24
Format 9-24

SUBSCHEMAS 9-25

Function 9-25
Format 9-25

TAPE 9-26

Function 9-26
Format 9-26

APPENDIX A PERFORMANCE NOTES A-1

APPENDIX B MAXIMUMS B-1

ILLUSTRATIONS

Figure	Title	Page
1-1	PRIME DBMS Documentation	1-2
1-2	Database Documentation Releases	1-4
2-1	Hardware Ring Structure	2-2
3-1	DBMS Layout	3-2
3-2	Area Format	3-3
3-3	Set Structure	3-5

FOREWORD

The Database Administrator User's Guide was written to aid the Database Administrator in determining what resources are available, how Prime's DBMS is structured and the methods by which a database can be created and maintained. This User's Guide serves three purposes:

1. to describe Prime's DBMS product for the prospective user,
2. to describe the Database Administrator's Command Processor, and
3. to provide instructions on database maintenance.

DOCUMENTATION EXCELLENCE

Prime is striving to maintain the highest documentation standards. To achieve this goal, the Database documentation will be published in three documentation releases as described in section 1. This is the Initial Documentation Release. Prime asks that each serious Database user correspond his comments about this manual concerning technical accuracy and additional information needed to implement the task of Database Administrator.

Robert E. Dawes, Technical Writer
Technical Publications Department
Prime Computer Inc.
145 Pennsylvania Avenue,
Framingham, Ma. 01701

ACKNOWLEDGEMENT

Prime Computer, Inc. wishes to formally acknowledge the work of the CODASYL Programming Language Committee (PLC) and the Data Description Language Committee (DDLC). The Data Base Task Group (DBTG) of the PLC produced in April, 1971 a report containing the specifications of a standardized data base management facility consisting of a Data Description Language for describing a database, a Data Description Language for describing that part of the database known to a COBOL program, and a Data Manipulation Language for COBOL.

The Prime DBMS, portions of which are described in this manual, is based almost completely on the April 1971 DBTG specifications. Prime Computer is also participating in the ongoing work of CODASYL in the area of data base management through its membership on the DDLC and on the Data Base Language Task Group (DBLTG) of PLC.

SECTION 1

INTRODUCTION

ABOUT THIS MANUAL

This manual is oriented toward knowledgeable Database Management System (DBMS) personnel (i.e., the potential Database Administrator or consulting expert) in Data Management. Readers are invited to make a direct comparison of the functional capabilities provided with the DBTG-71 specification.

The reader is assumed to be acquainted with the basic concepts of Virtual Memory Operating Systems; he should also be intimately familiar with Data Management in general, the benefits of Database Management in particular, the DBTG-71 Report, and relevant Prime product bulletins.

PRIME DBMS DOCUMENTATION

DBMS documentation (Figure 1-1) is provided for both the Database Administrator and the application programmer. The Database Administrator uses two manuals: 1) The Prime Database Administator User's Guide and 2) The Prime DBMS SCHEMA Data Description Language (DDL) REFERENCE Manual.

The application programmer uses two manuals per application language: 1) the Prime FORTRAN Reference Manual for DBMS and the companion Prime FORTRAN User's Guide; 2) The Prime COBOL Reference Manuals for DBMS and the companion Prime COBOL User's Guide.

The Prime DBMS SCHEMA DDL Reference Manual contains definitions and rules for using the DDL syntax and "How to Use" procedures for the Schema DDL Compiler.

The application programmer uses the COBOL and FORTRAN Reference Manuals for DBMS for syntax definition and rules and "How to Use" procedures for the Subschema DDL Compilers and DML Preprocessors.

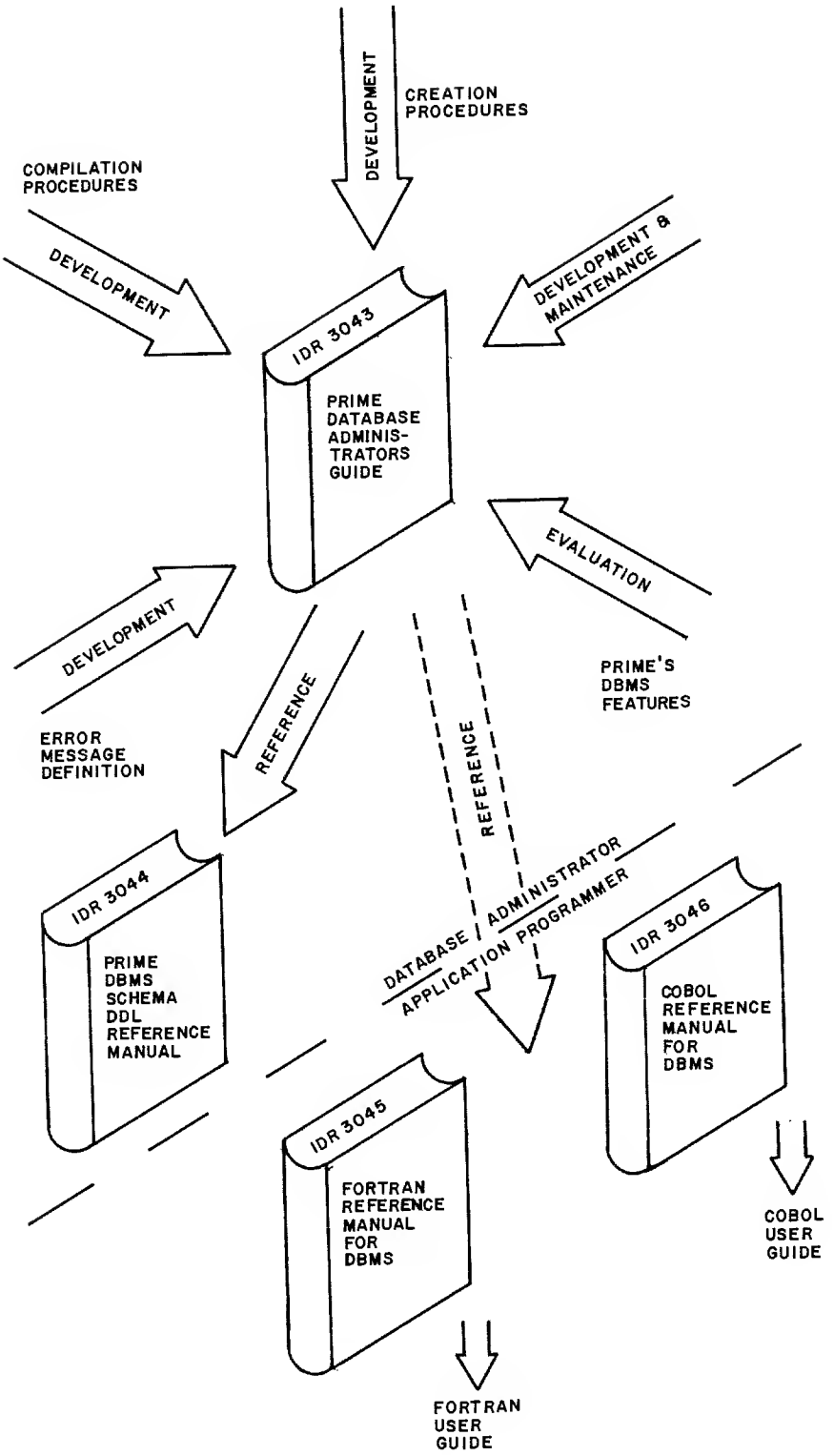


Figure 1-1. PRIME DBMS Documentation

Documentation Releases

Prime provides three documentation releases (see Figure 1-2) for every new product: The Initial Documentation Release (IDR), the Preliminary Documentation Release (PDR), and the Final Documentation Release (FDR).

The Initial Documentation Release (IDR) provides advanced information. The intent is to provide usable, accurate information without regard to style and format.

The Preliminary Documentation Release (PDR) is the second draft by the writer. It provides more complete and accurate information about the product, but does not represent the final document format.

The Final Documentation Release (FDR) is the complete product description up to the stated software revision number. This Release is edited, formatted and presented in Prime's highest professional standards. Users will be notified when this release is available.

OTHER RELATED PRIME DOCUMENTS

- o PRIMOS FILE SYSTEM USER GUIDE (MAN26Ø4)
- o PRIMOS INTERACTIVE USER GUIDE (MAN26Ø2)
- o PRIMOS COMPUTER ROOM USER GUIDE (MAN26Ø3)
- o PROGRAM DEVELOPMENT SOFTWARE USER GUIDE (MAN18790)
- o FORTRAN IV USER'S GUIDE (MAN3Ø57)
- o COBOL USER'S GUIDE (MAN2797)

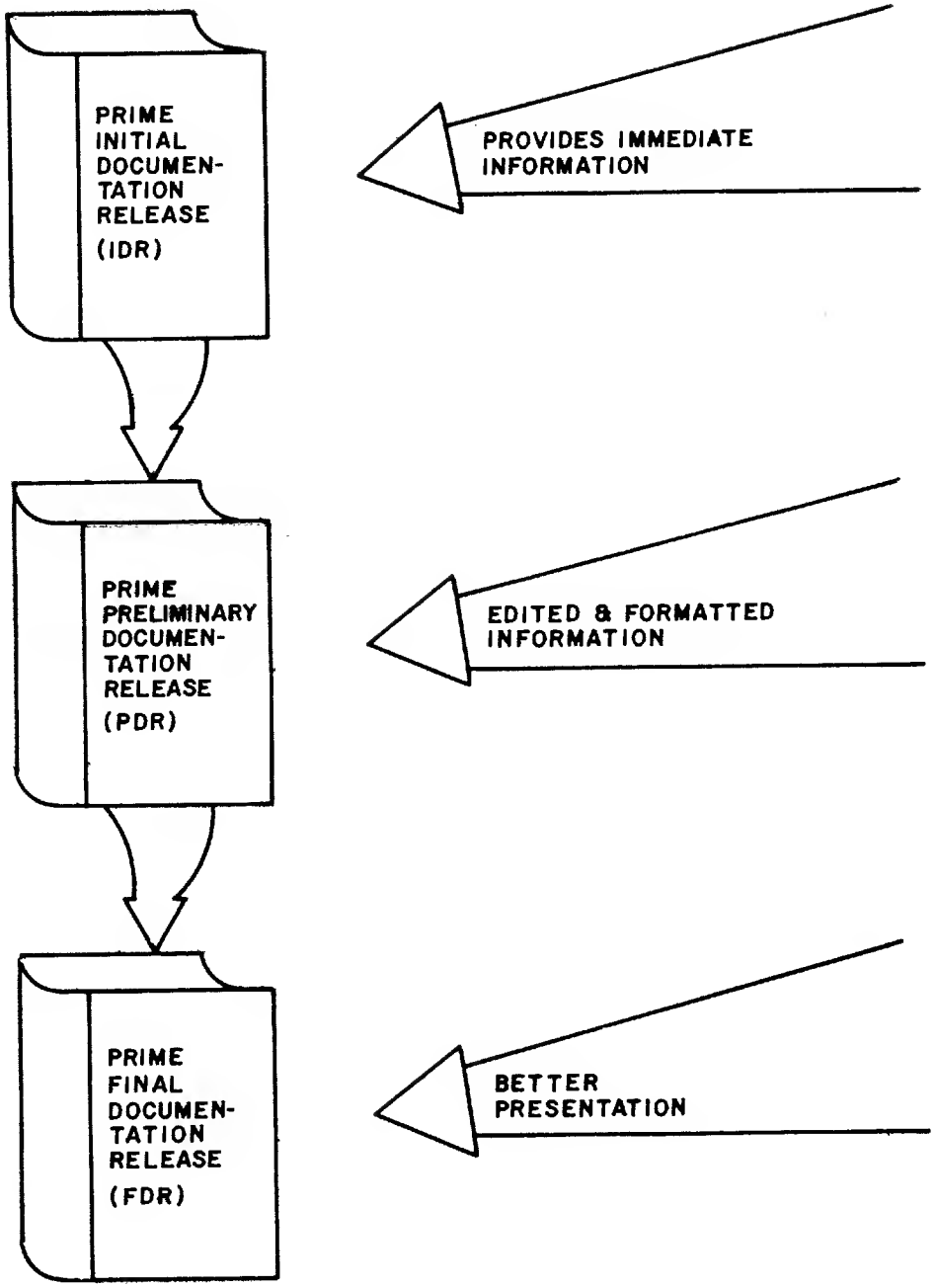


Figure 1-2. Database Documentation Releases

LIST OF DBMS REFERENCE DOCUMENTS

This document is written assuming that the reader has a certain level of knowledge concerning the concepts and environment relevant to the Prime DBMS. To define the prerequisite level of understanding, the following list of documents serves as a reference. The required reading list includes some information on the PRIMOS system as well as the n CODASYL (DBTG-71) report.

Required Reading:

1. CODASYL Database Task Group April 1971 Report (DBTG-71)

2. "PRIMOS IV AND V", Prime Product Bulletin

3. "Database Management System (DBMS)", Prime Product Bulletin

Suggested Reading:

1. DBMS Concepts: (either of the following, or equivalent)

   "Database Systems: A Practical Reference" - Ian Palmer
   QED Information Sciences
   Wellesley, Mass. 1975

   "An Introduction to Database Systems" - C. J. Date.
   Addison-Wesley, Reading, Mass., 1975

   "Computer Database Organization" - James Martin.
   Prentice Hall, Englewood Cliffs, N.J., 1975

2. Balanced Trees and B-Tree Variations: Set Implementation Technique

   "Sorting and Searching" - D. E. Knuth
   "The Art of Computer Programming, Volume III"
   Addison-Wesley, Reading, Mass., 1975

3. CODASYL Systems:

   CODASYL Database
   Management Systems,
   ACM Computing Surveys
   Vol. 8, No. 1, March 1976

SECTION 2

PRIME'S DBMS ENVIRONMENT

INTRODUCTION

This section focuses on the contribution of the environment provided by the Prime hardware and PRIMOS software to the achievement of the design objectives of the DBMS.

HARDWARE

Prime's DBMS operates on a Prime 400 system or higher. The Prime 400 and 500 are true virtual memory machines with a large address space (512 Megabytes per user for up to 63 users). While they are designed specifically for time-sharing applications, any multi-user environment benefits from their memory management (for example, transaction processing).

In addition to the multi-user environment capability, the hardware contributes significantly to the speed, integrity, security, and upgrade flexibility of the entire system.

Speed

Hardware design has aided overall speed in a Prime system. Instruction execution speed is greatly enhanced by utilization of a cache memory which remembers data from recent memory references. Subsequent instructions referring to the same locations actually get their data from the cache. This in effect turns those memory reference instructions into register reference instructions, for a speed enhancement of approximately 3:1.

Subroutine Calls: A direct hardware implementation for handling subroutine calls and passing arguments significantly reduces the overhead involved with calling subroutines. This is extremely advantageous to systems such as DBMS where subroutine calling is heavy.

Paging Speed: Paging speed has been enhanced by the new disk storage module because its record size is identical to the central processor memory page size; this simplifies buffering and paging.

Page Size: The large page size, 2K bytes, increases the likelihood that desired information will be in high-speed memory since instructions and data that are used together frequently occur together, e.g., the contents of a record. Furthermore, the implementation of the storage structure of the DBMS has carefully taken advantage of this large page size.

Integrity and Security

Integrity and security is an important aspect of database system requirements. Prime's hardware environment includes internal transmission checking and a ring structure.

Prime CPU memory has always been highly reliable because Prime hardware performs parity checks at both ends of every internal transmission. The memory for the Prime 400 and 500 machines is built in an error-correcting array: no word can be incorrectly transmitted except by multiple failure.

Contributing to the integrity of the system is enhanced security. The hardware is ring-structured (see Figure 2-1) to protect software and data. This ring structure enforces security by checking every memory address reference at the time it is generated by the hardware, and trapping unauthorized references to any lower-level ring.

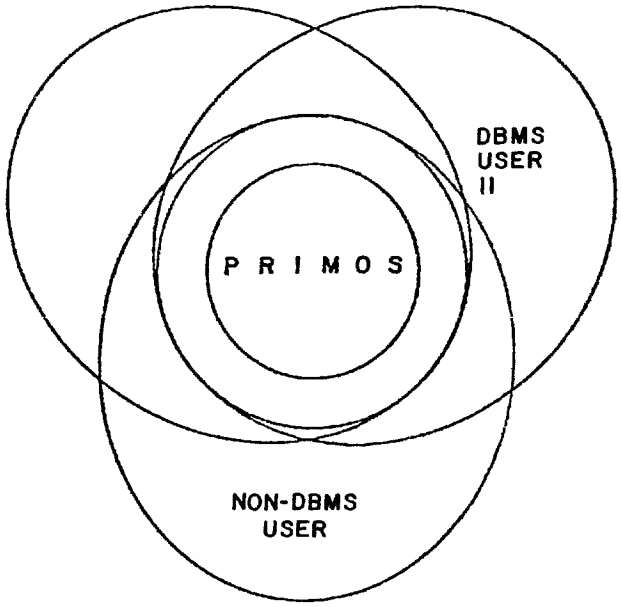


Figure 2-1. Hardware Ring Structure

PRIMOS resides in the innermost ring, where it is accessible only via legitimate calls. The DBMS resides in a ring of its own, just outside the PRIMOS ring. User programs reside outside the PRIMOS and DBMS rings and consequently cannot access anything in those rings (for example, password tables) except via legitimate function calls (which are examined for proper authority). Any unauthorized attempt to READ, EXECUTE, or WRITE in any ring from outside causes the offender to be trapped by the hardware and aborted.

The Prime storage module disk controller uses a sophisticated polynomial algorithm capable of detecting error bursts exceeding 30 bits, and correcting bursts up to 11 bits in length. This means that transient failures which are largely fast intermittent errors, are virtually eliminated. In other words, unrecoverable disk errors rarely happen anymore, except on disks that are actually damaged.

Hardware Upgrade and Expansion

Prime has designed an upgrade and expansion capability into all Prime machines.

The procedure for upgrading the CPU of a Prime machine to a Prime 400 or higher consists solely of replacing the current boards with different ones.

Similarly, the procedure for expanding high-speed memory consists solely of plugging in an additional memory board or replacing an older/smaller/slower board with an improved one. PRIMOS adjusts completely and automatically to its new environment.

Adding more disk memory requires only the additional step of altering the PRIMOS configuration table to reflect the addition. A command exists to perform this function (see Section 9).

The architecture of the DBMS reflects this ease of expansion in its treatment of areas, allowing a database to expand naturally as new disk facilities become available.

PRIMOS ARCHITECTURE

The flexible architecture of the Prime hardware/software complex provides a substantial base for the DBMS.

PRIMOS (Prime Operating System) itself has a great deal to offer the DBMS. Most obvious is its orientation toward multiple-users. Beyond that includes the provision of device independence. PRIMOS offers a system in which a user can simply log in and process files with virtually no concern for physical protocol, regardless of his approach (interactive or not) or the services he desires (DBMS or not, etc.).

File and Access Management

PRIMOS also contributes the services of its File Management System (FMS). PRIMOS interfaces to DBMS via a Random Access Manager (RAM), provides utilities and an optimizing ANS FORTRAN IV Compiler - the implementation language of Prime's DBMS. These software components make significant contributions to the speed and integrity of the DBMS, provide a foundation for its data structure, and add some lower-level capabilities.

FMS is used as the undercarriage of the DBMS for its physical I/O management. RAM provides the interface to DBMS, which accomplishes logical file management.

RAM multiplexes the use of logical file units, thereby extending the concept of "virtualization" to files. This allows more files to be open than there are file units to use, thereby rendering these physical restrictions invisible to the user.

(The word "file" has no defined usage in the DBTG-71 Report, and consequently is reserved by Prime to mean the physical entities upon which the FMS operates.)

Integrity in PRIMOS

Another contribution by PRIMOS is its built-in integrity. FMS threads the physical blocks of its files bi-directionally. A utility (FIXRAT) is provided to examine and repair file faults using this threading.

RAM accomplishes concurrent update access via its total control over the buffer pool, where it provides for multi-threading of user requests on the DBMS I/O and supports database recovery procedures.

Security within PRIMOS

Security is provided by PRIMOS through its access rights. FMS provides dual password protection at the physical file level: one password designated as "Owner" and another as "Non-Owner". These are each associated with different access rights.

FMS provides different types of access rights:

No-Access	Delete-Only
Read-Only	Delete-Truncate-Read
Write-Only	Delete-Truncate-Write
Read-Write	Full-Access

Buffer Pooling

Another PRIMOS feature is the efficient memory management. FMS provides physical blocking and buffer-pooling, for efficient memory management at the data level (as opposed to memory management at the instruction level; i.e., virtualization).

RAM provides logical buffer-pooling, the sharing of common files and distribution of data to requesting users. It selects its buffers using a Least-Recently-Used algorithm.

This architecture provides cascaded levels of memory management, each adding advantages of its own, and building on the previous level. Virtualization is extended in both directions from the page level (PRIMOS) to the file level at one end (by RAM) and the word level at the other (by CPU cache), while two levels of buffer pooling provide physical (FMS) and logical (RAM) sharing.

Data Structure Assistance

PRIMOS construction and utilization of segment directories, essentially a file directory at a physical level, provide a sound foundation (with FMS) for DBMS file structure.

FMS provides direct access files which DBMS utilizes to implement its data structure, simultaneously profiting from the other FMS benefits mentioned above.

Speed

The newest and most crucial feature of PRIMOS to the DBMS is its ability to share common program space with multiple users. All DBMS code is re-entrant, and is therefore shared single-copy code, but PRIMOS simultaneously treats it as though it were a user-bound subroutine. Thus the advantages of shared code (minimizing paging) and user calls (no interrupts with low-overhead parameterization are both realized).

Among the single-copy items shared by the DBMS, besides PRIMOS and DBMS itself, are the object schema (database model), the buffer pool, and tables of locks for controlling concurrent use.

Maintainability

PRIMOS facilities include its optimizing ANS FORTRAN Compiler, allowing the entire DBMS to be written in a higher-level language. This enhances its maintainability, yet the optimization minimizes the execution price of using a higher level language.

DBMS ARCHITECTURE

DBMS has been designed to place independent features into separate independent components, as enumerated below:

1. Schema Data Definition Language (Schema DDL) Compiler
2. COBOL Subschema Data Definition Language (Subschema DDL) Compiler

3. FORTRAN Subschema Data Definition Language (Subschema DDL) Compiler

4. COBOL Data Manipulation Language Pre-Processor

5. FORTRAN Data Manipulation Language Pre-Processor

6. Database Administrator Command Processor (DBACP)

It is especially important to note that this separation of components permits definition of databases, generation of databases, and definition of private user views of these databases (subschema) to proceed independently, each in the language most suitable for it. The data manipulation commands are pre-processed to produce ANS COBOL or ANS FORTRAN final versions of the application program.

The DBMS has also made use of modularity in that it has re-used techniques wherever legitimate (for example, set ordering and search keys share a single high-speed technique; database generation and database expansion are virtually identical). Additionally, many low-level modules are used in more than one of the above components.

SECTION 3

STORAGE STRUCTURE SUPPORT

INTRODUCTION

This section describes some features of PRIME's DBMS storage structure in terms of database layout, area layout, set format and search keys.

The DBMS is capable of handling many different databases, each with a structure which may be as complicated as a full network.

Although each DBA controls the databases at his installation, the existence of passwords for all schema operations provides for a separation of authority.

Database Layout

The DBMS layout is illustrated in Figure 3-1. Multiple databases imply the need for a schema directory. Each entry in the schema directory holds an internal identifying number and a pointer to a table which holds information about the database. (schema table). The schema table includes a subschema directory (which lists the subschema associated with this schema) and definition tables which define this database with respect to space on physical volumes, and all information contained in the schema DDL.

The set and record definitions are the logical view of the database and the area definitions are the physical view. The implementation of their relationship is naturally a crucial determinant of the effectiveness of the entire system.

Area Format

The Area definition is illustrated in Figure 3-2. Each area is divided into buckets. The buckets are chosen to be multiples of physical block size appropriate to house the specific record types assigned to them. Because of the relatively large page size (and its correspondence to block size), this multiple is often 1, which contributes greatly to the effectiveness of paging in data records.

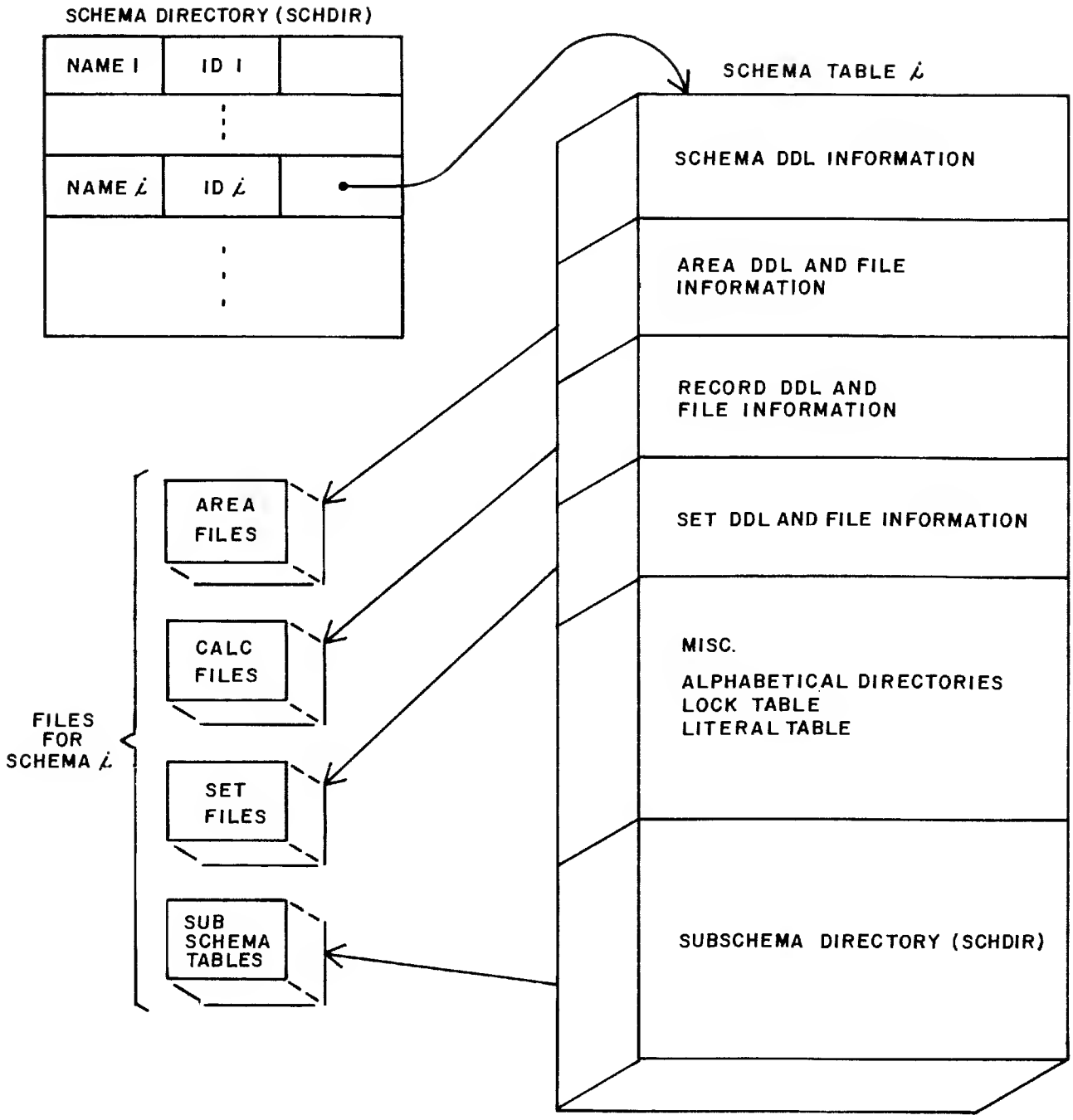


Figure 3-1 DBMS Layout.

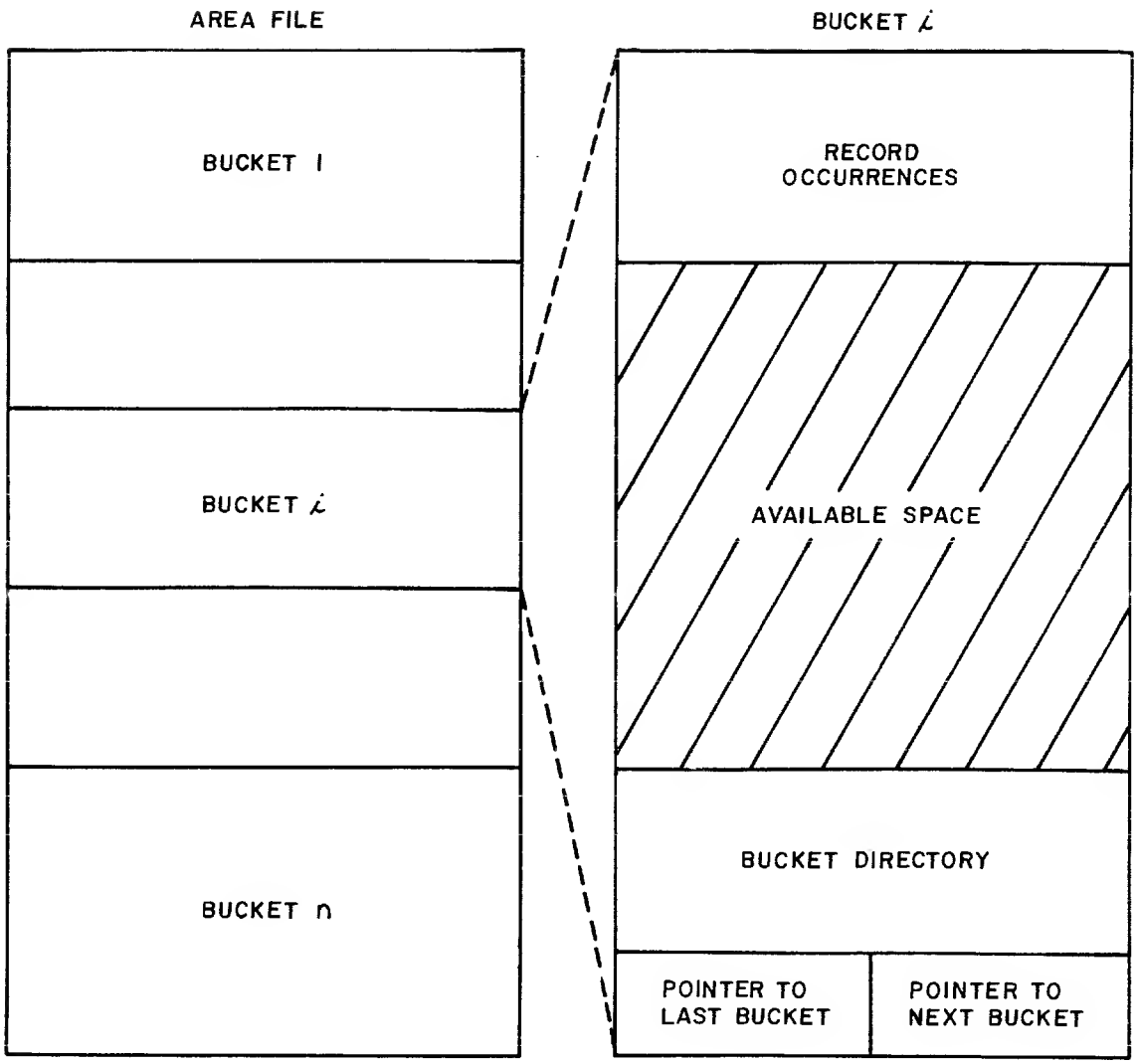


Figure 3-2 Area Format.

Each bucket has its data records built down from the top with a directory at the bottom. If necessary, any data record may span many buckets, but no matter how it expands or contracts, it will always be located through the bucket to which it was originally assigned.

Each record's database key is a pointer to it built of several portions: 1) the identity of the area within the area definition for the database (up to 1K areas/databases), 2) the bucket within the area (up to 1M buckets/area), and 3) the directory entry within the bucket (up to 255 entries/bucket). The key also contains the record type identifier (up to 1K/database). Since the record never changes bucket directories, the invariance of the database key is preserved.

Set Format & Search Keys

The Set structure is illustrated in Figure 3-3. Sets are implemented via external multi-level pointer arrays which are a variation of "B-trees" (as described by Knuth). The leaves of the B-trees are database keys. The nodes of the B-trees are indices of the next lower level in the B-tree.

The Prime B-tree variation has several advantages. First, the nodes are doubly-threaded (left and right) for fast traverse. Second, the node size (fanout) is chosen on a type of set basis to reflect the average significance of database key occurrences for the type of set. Third, the association of nodes with page size leads to quicker access.

A separate B-tree is also built for each search key of a set. Thus the B-tree mechanism does double-duty as a fast data inversion method.

The entry for each owner occurrence in the set directory contains a pointer to the head of the B-tree relevant to that occurrence.

To enhance traverse, the nodes of the B-tree are not only threaded bi-directionally, but point back to the parent node, aiding in insertion and removal of member entries.

Any database may span any number of available volumes, even though the DBMS I/O is founded on the FMS (which cannot span volumes). This is accomplished by restricting the files containing types of sets, calc tables and data base areas to a partition of a volume, while allowing different files to reside on any available volume.

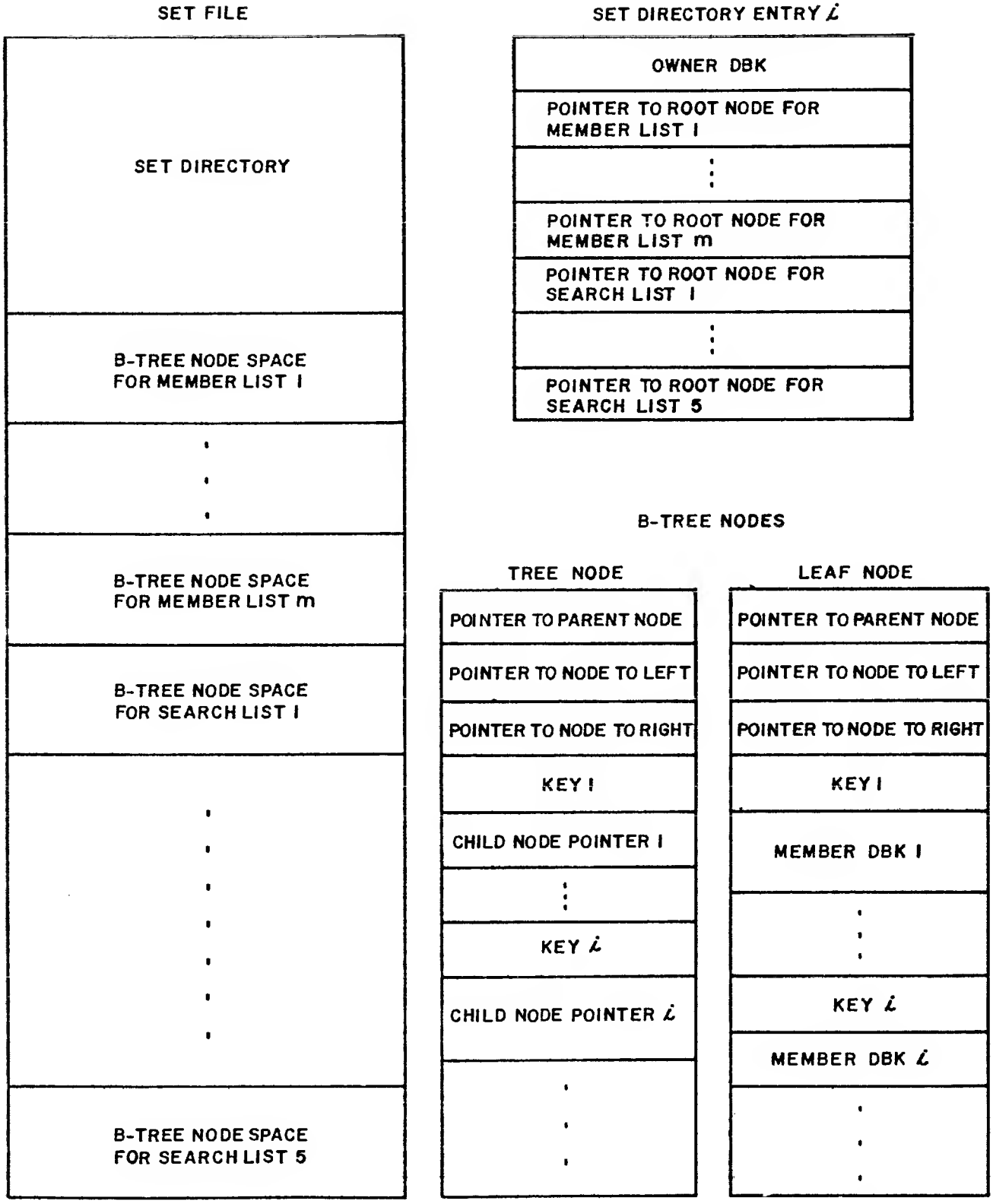


Figure 3-3 Set Structure.

Given the orientation of the DBMS toward rapid access, one would expect speed to be a primary objective or the implementation approach. B-trees have demonstrable advantages for retrieval, update, and deletion when compared to other data structures involving ordered sets.

B-trees themselves are tunable (by varing fanout) for measuring space-time tradeoffs. Consequently, Prime has chosen to use B-trees exclusively, which adds uniformity to the efficiency of this approach.

LOCATION Mode

Location mode CALC, is implemented as a hashing algorithm into a table of database keys. Collisions are resolved by a threaded list overflow technique which links all calc records having the same hash address together. Record occurrences with identical CALC keys (DUPLICATES) are also threaded.

Placement of the data is of limited concern to a Prime DBMS Database Administrator (DBA) because of the Prime hardware optimizing random-sectoring technique; proximity other than same page is meaningless. To take advantage of the benefits of page-proximity, the DBA can designate record-types in sets to have a location mode which is via set. The DBMS then stores these member record occurrences as close to their owner occurrence as possible.

If no location mode is specified, a record is stored in the next bucket with available space.

Ease of Expansion

A major concern of the Database Administrator is the complexity of expansion. PRIME's DBMS is expandable through the following techniques:

Areas can be expanded to include more buckets. Similarly, bucket sizes may be expanded. The DBMS automatically pads the extra space. A record re-organization may be requested at this time (PACK) to use the new space as efficiently as possible.

Areas which contain no data can be initially defined for later expansion. This makes it possible to define a database which is going to grow substantially over the years using today's sizes and today's storage space, and to simply expand old areas onto new devices, as the size of the data demands and the acquisition of more storage space permits.

The use of buckets also highly localizes the effects of record alteration. Records can expand and contract with little effect upon other records (except for bucket-mates). If new space is needed by an expanding record, a new bucket is acquired for it to flow into. A free-bucket chain is kept to minimize search time for new buckets. This whole process obviates the need for special overflow mechanisms,

and minimizes the price of "overflow accesses" paid by the physical system. Garbage collection occurs only upon the request of the DBA so that update users pay no execution penalty for alteration of record size, unless the accumulation of alterations affect system performance.

The fixing of the database key, the logical consistency and the saving of space more than cancel out the extra access needed to reach an extension of a record into another bucket.

Space Utilization

Field experience suggests that total space overhead ranges from approximately 5 to 6Ø percent, the latter occurring when there is a great deal of structure. Yet, such files have generally been found to take no more space than their pre-DBMS counterparts, which implies that the actual cost of this overhead approaches zero. The reason for this is simply that the redundancy required by non-DBMS systems requires as much space as Prime DBMS overhead. But there is a large improvement in capability and flexibility (as in access speed) obtained by utilizing that same space for DBMS-structured data.

SECTION 4

THE PRIME VIEW OF SOME COMMON DBMS CONCEPTS

INTRODUCTION

This section discusses Prime's approach to implementing many common considerations. These include:

- o DATA STRUCTURES
- o ACCESS STRATEGIES
- o DATA INDEPENDENCE
- o PRIVACY
- o INTEGRITY
- o SHARING/CONCURRENT UPDATE PROTECTION
- o PRESERVING THE PHYSICAL INTEGRITY OF THE DATABASE
- o RECOVERY
- o DATA BASE ADMINISTRATOR SUPPORT

Data Structures

All data structures recommended by DBTG-71 are supported. These include ordered, tree, hierarchical, cyclic, and network. However, as per DBTG, no two record types may own each other directly; a third record type must be invented to own them both, thereby associating them as desired.

Access Strategies

The supported access strategies have been partially discussed in earlier sections. There are access-via-set capabilities provided via the set directory with linked-to-owner feature; there are indexed-entry capabilities in several forms: via a search key as provided by the B-trees or via a search key provided in the same manner, or via the CALC function, hashing values to scatter table containing database keys. There is also direct access via the database key.

Data Independence

The data independence provided by the DBTG is realized by the definition of subschemas and their relationship to the schema. The DBTG states that a Subschema Data Definition Language (DDL) and a Data Manipulation Language (DML) should exist for each application program language which will interface to the database, but DBTG-71 considers only COBOL.

Prime has implemented the subschema Data Definition Languages (DDL) and Data Manipulation Languages for both COBOL and FORTRAN. While the manipulation languages are nearly identical and very English-like, as the DBTG recommends, the Subschema Data Definition Languages reflect their host languages quite clearly.

COBOL and FORTRAN recognize different data types and have different naming conventions. Their respective DDL's address themselves to the appropriate data types for each language. Since implicit data typing is permitted by FORTRAN, it is also permitted by the FORTRAN DDL: the implicit data type is that of the schema (or an appropriate default variation if that type does not exist in FORTRAN).

During compilation, the User Work Area is constructed as a COMMON Block for FORTRAN and as WORKING STORAGE declarations in COBOL. A tabular listing of the User Work Area is provided by the subschema compilers, thus giving the user a complete definition of his work area, including all the items and their types.

All schemas and subschemas are compiled into an object table for faster reference. Hence, whenever a schema changes, it and all its subschemas must be recompiled. Of course, if a subschema is changed, it must be recompiled.

Changes to subschemas affect all application programs using them, which must generally be recompiled. All changes to schema currently require such recompilation, since all subschema are affected.

Were this not true, the enterprise would be faced with a particularly dangerous form of data independence. Changes which affect the User Work Area imply that run-units using that work area are now outmoded. They must at least be checked; particularly if the definition of the database has been changed, or the view of it represented by a run-unit's subschema has been changed in previously existing run-units. However, if an older version of a subschema is deleted, run-units referring to it will be trapped by the DBMS. This represents an enhancement to the potential integrity of the database.

The Prime DBMS provides an entirely new level of data independence called "dynamic reference", based on really not caring about the actual data contents. This is the type of data independence which general text editors, dump routines, and other utilities appear to have. DBMS achieves this by delaying binding until run-time. The schema and subschema to be used by the run-unit need not be declared until

run-time. Thus generic utilities which view the schema through any subschema can be written. This capability is described further in Section 5, Extensions.

Privacy

The natural attributes of the hardware and software allow an effective Privacy capability. Privacy is maintained well beyond the capability set down by the DBTG specification. The PRIMOS Password and Protect capabilities are used here. The Hardware Ring Structure provides a measure of security unachievable in any other fashion. Lists of DBAs and their privacy keys may be kept in memory and be absolutely inaccessible to unauthorized personnel.

For further protection, keys are supplied at run time. This means that access to a source listing is not access to the keys it must use - for the key can be supplied interactively at run-time.

As recommended by DBTG, every operation on every DBMS entity can carry a separate lock. These locks may be literal or variable. Those which are declared as variable can be changed by the DBA at his discretion.

Integrity

The Prime DBMS provides integrity in many ways already mentioned. The hardware provides parity checks at both ends of all gating paths, the CPU memory and disk controller both utilize error correction algorithms. The FMS provides bi-directional threading of all files and a utility to provide file maintenance. RAM provides control over logical sharing of the data.

The DBMS also prepares extensively for recovery of data in event of failure, including before-imaging, after-image journalling, and database transaction-oriented delayed incremental updates.

Sharing/Concurrent Access with Most DBMSs

In most database systems, concurrent access to data present no problems as long as data does not change. If any users change shared data, several problems arise.

Insuring logical consistency, when shared has changed: If any user is changing shared data while one user is attempting to analyze the data, the analysis may be meaningless. Logical inconsistencies may flaw the analysis since the analysis does not represent a single state of the data.

For example, suppose that user A is reading a personnel database to produce a list of employees by department. If another user transfers employee X from a department that has been listed already to a department that has not yet been listed, the employee X will appear to be a full-time employee to both departments. Thus, user A has an inconsistent view of the data.

Lost updates when shared data has changed: Suppose that the following sequence of events occurs:

```
user A reads item X
user B reads item X
user A replaces X by X + 1Ø
user B replaces X by X + 2Ø
```

Notice that the update by user A has been lost. This occurs because user B read the same value of X as user A rather than reading the result of the update by user A.

Locks are often employed to eliminate the above problems. Parts of files are locked while they are being analyzed or updated. Not only does this raise the problem of lock management, but it can lead to a situation known as deadlock. Consider the following sequence of events:

```
user A locks record X
user B locks record Y
user A requests record Y
user B requests record X
```

User A waits on user B, while user B waits on user A. Neither user can proceed. One user must be aborted. How can a decision be made as to which user to abort? What should be done about any records that have been modified by the user that is selected for abortion?

Sharing/Concurrent Access with Prime's DBMS

The Prime DBMS eliminates the problems of inconsistent analysis, lost updates, deadlock, and many other problems resulting from concurrent update of files by introduction of Database Transactions (DBTs). DBTs are used to group data accesses which must be performed together to insure consistency of a database. A DBT is introduced by a START TRANSACTION DML command and is terminated by an END TRANSACTION command or an ABORT TRANSACTION.

There are two types of DBTs: Retrieval DBTs and Update DBTs. The two types of DBT provide different sets of features.

The Retrieval DBT provides the user with a consistent view of a database even though concurrent users may be updating the database. This is done by saving before-image copies of modified blocks. When a Retrieval DBT requests information from a modified block, it is given the before-image copy of the block as it existed when the Retrieval DBT started. Thus, all updates initiated since the beginning of the retrieval transaction are transparent to the Retrieval DBT.

The Update DBT provides a consistent view, prevents lost updates, prevents deadlock, and provides rollback of updates. As an Update DBT modifies blocks, before-image copies of the blocks are saved. The

before-images are used to rollback a transaction when a user aborts the transaction or when a transaction is aborted automatically.

An Update DBT is not permitted to read or write a block that has been modified by a concurrent Update DBT. If this restriction is violated, the offending transaction is notified that it must abort. The transaction can perform its own recovery before aborting the transaction. The transaction will be aborted automatically if it attempts to execute any DML command other than ABORT TRANSACTION.

Since concurrent Update DBTs are permitted to proceed only if they operate on different subsets of the database, a consistent view of the data is insured, and lost updates are eliminated. Transaction rollback provides recovery from deadlock and recovery from many errors including DML program errors, certain DBMS errors, and even halts due to hardware or software errors.

When an Update DBT aborts because of concurrency conflict, the program can immediately begin another DBT to attempt the same update. Concurrent update conflicts are transient and usually clear very quickly.

The key elements in the implementation of DBTs are update transaction numbers, lists of completed transactions, and before-images. When an Update DBT starts, it is given a new transaction number and a list of transactions that have completed. An Update DBT is only permitted to read and write blocks that were last written by transactions that are in the list of completed transactions for the Update DBT. Whenever an Update DBT modifies a block, its transaction number is written into the header of the block. Before the modification is done, a before-image copy of the block is saved in a before-image file. The before-image and the modified block are linked together for use in transaction rollback, other recovery procedures, and Retrieval DBTs.

When a Retrieval DBT starts, it too, is given a list of completed transactions. If the Retrieval DBT attempts to read a block that was written by an incomplete transaction, the chain of before-images for the block is searched backward for the most recent block that may be read by the Retrieval DBT.

By using Update DBTs, the blocks dynamically involved in an update are inaccessible for other updates. An exception occurs only when one Updated DBT actually attempts to access data which has been modified by another concurrent Update DBT. In addition, Retrieval DBTs never lock out Update DBTs, yet consistent data is always available to Retrieval DBTs in the before-images. The use of DBTs results in a much smaller fraction of the database being locked than schemes which require pre-locking of data to be updated to be retrieved. In particular, portions of the database may be dumped while others are concurrently running against those same portions, since a dump program can be programmed as a single retrieval DBT!

Preserving the Physical Integrity of the Database

In the overwhelming majority of cases, each DBT will be interested in a very few records for a very short time. No conflicts of any sort will occur, and this entire system will be utterly transparent, because the new version of each record replaces the old in the very same spot and no additional accesses of any sort are required.

The only inconsistencies that can creep into the data base under this system are errors introduced by application programs which are authorized to make changes and do so incorrectly (i.e., faulty user program logic).

Operations which must affect significant portions of the database are performed without concurrency at the request of the DBA, which is effected by locking the entire schema against any access. The operations are:

Command	Function
ALLOCATE	Allocate space for the files of the database and preformat it.
EXPAND	Allocate more space for the files of the database.
PACK	Garbage collect and reorganize the specified areas/calc files.
SAVE/RESTORE	Copy the database to/from magnetic tape.

NOTE:

These facilities are with the DBACP and are described in Section 9.

RECOVERY

Many kinds of errors, exceptional conditions, and failures can occur in the use of a DBMS. A sophisticated DBMS must provide a variety of recovery techniques to assure the reliability of the system and the integrity of the data.

The Operation of the Run-Unit

The DML Command Process (DMLCP) performs extensive validation of each DML command issued by a run-unit. The DML languages provide facilities for a DML program to trap all nonfatal errors in two ways. Each DML command may include a list of errors that the program wants to trap and where to go if an error is detected. A run-unit can also check the error status in-line after each DML command and perform the appropriate

recovery if necessary. When the DMLCP detects an error during the execution of an UPDATE DML command, it rolls back the database to its status before the erroneous DML command. The run-unit is given the opportunity to recover from the error. If another DML command is attempted before the error is cleared, the DMLCP aborts the run-unit and automatically rolls back the entire transaction (if one is active).

Roll Back

Several kinds of exceptional conditions can occur during concurrent update of files. The most frequent exception is when concurrent transactions attempt to update the same block. Concurrent update exceptions can be trapped by the user in the same manner as other DML errors. After a concurrent update exception has occurred, the DMLCP will always roll back the transaction when the next ABORT TRANSACTION command is executed. If the DML program detects an error in any way during a transaction, it can execute an ABORT TRANSACTION command, and the DMLCP will roll back the transaction.

The Recovery Processor

If a DML program does not terminate properly, it can leave the database in an indeterminant state. Termination can be caused by the program user hitting the break key, by the DML program exiting before ending a transaction, closing the areas, and exiting the DBMS, by fatal errors in the DML program, the DMLCP, or system software (errors such as division by zero, invalid addressing, end of file, etc.), or by a system halt due to hardware or software failure. The Prime DBMS includes a recovery processor. The recovery processor includes commands to report the status of transactions and DBMS usage and to roll back all incomplete transactions and to clean up DBMS control tables.

Sometimes a DML program that appears perfectly correct to the DMLCP introduces errors into a database. Perhaps the DML program deleted or modified the wrong record; it may have deleted the entire database. It is also possible that a new version of the DMLCP could contain some bugs that introduce errors into a database.

Recovery Facilities

The Prime DBMS provides a couple of facilities for recovering from the errors mentioned above and other catastrophic errors. Before a block in the database is modified, a before-image copy is saved in a before-image file. A modified block is written out to disk, a duplicate copy is written to the after-image file. All blocks are labeled by the transaction that last modified the block. In addition, a message is written into a log file whenever a transaction starts, ends, or aborts, and whenever a block is modified. Before-imaging, after-imaging, and logging can be turned on and off separately by using the Database Administrator's Command Processor (DBACP), see Section 9.

The I/O subsystem of the DBMS has been designed to insure that the

recovery files are robust to system halts and other failures.

The recovery processor can examine the log file to help identify faulty transactions and to analyze update activity since the faulty transactions. The recovery processor can roll back the database using the before-image file or roll the database forward using a saved copy of the database and the after image file. A copy of a database can be saved and restored by use of PRIMOS MAGSAV/MAGRST or through use of SAVE/RESTORE in the Database Administrator's Module. The selection of rollback or rollforward depends on the size of the database, on the volume of updates since the last database save, on the volume of updates since the errors, and on the type of failure.

Analysis of the log file may sometimes indicate that a restore and rollforward of part of the database will still guarantee integrity of the data and speed up the recovery process. The recovery processor includes features to expedite this.

Recovery from a disk head crash can be done by restoring the affected areas from a database save and rolling forward using after-images.

DATABASE ADMINISTRATOR SUPPORT

Privacy-Locking

Each installation's databases are overseen by Database Administrators (DBA). Each DBA potentially has control over all databases at installation. However, a database may be removed from a DBA's authority by Privacy-Locking its usage, thus barring those DBA's who are not authorized to know its keys.

Each installation may define to the DBMS how many of the DBA's (including none) will be privileged DBA's. These DBA's are understood by the DBMS to know all necessary keys, and thus have special privileges useful in database debugging and assisting other DBA's and applications programmers.

Caretaker Functions

The function of the DBA is essentially that of caretaker of the database. The DBA creates the model, populates it, monitors it, tinkers with it, revises it, improves it. Help is provided to the DBA in accomplishing this multifarious task by the Database Administrator Command Processor (DBACP).

Because the DBA functions are so far-ranging, each can be guarded by its own lock. In fact, since all the locks are single-purpose, an extensive dialog is often required for a DBA to accomplish his objectives. These safeguards are intended to protect against careless errors.

Creating the Database

The DBA component helps the DBA in a number of ways. The earliest function which must be provided is database creation.

Definition of the schema, it should be recalled here, is outside the realm of this component, since a separate component, the schema DDL Compiler, exists solely to perform this task.

Anyone may use the DDL to define, and even compile a schema; but only a DBA may allocate space to the schema and populate it with data. There is an advantage in this. A DBA can assemble a staff of technical experts, but can investigate the proper modeling of it, experiment with possible models, and ultimately determine the best one, technically, for the DBA to use.

SECTION 5

EXTENSIONS

INTRODUCTION

Most of the capabilities already discussed include functional and/or operational extensions to the DBTG-71 Report recommendations. This section focuses on the more important global extensions that have been mentioned, and some specific ones that have not.

GLOBAL EXTENSIONS

Database Transactions (DBT's)

The existence of DBT's and their value has been fairly extensively discussed. The existence of the START and EXIT/ABORT transaction block structure allows dynamic implicit locking of data instead of static pre-locking in concurrency situations.

INVOKE Command

The INVOKE command is executable. Prior to executing the INVOKE command, a run-unit is not associated with the DBMS. The EXIT/ABORT DBMS commands disconnect DBMS from the run-unit. DML commands occurring outside the scope of an INVOKE are ignored. Multiple INVOKE-EXIT/ABORT pairs are permitted, but all must reference the same subschema.

Keys

Keys are supplied interactively at run time for more privacy. It is also easy for a DBA to change keys for variable privacy locks.

ON ERROR Clause

An ON ERROR clause applicable to each DML command allows the application program to specify alternatives for each different type of possible error on that command, so the program can handle them in whatever fashion it chooses (barring required ABORT). The ON ERROR clause may specify either GO TO label or CALL external value.

Dynamic Reference

Dynamic reference provides total data independence for special purposes, by delaying binding to a schema/subschema until run time. The names of such database entities as sets, areas, record-types, and items may therefore be referenced dynamically, as conversational input via an interactive terminal, for example. An application program written to utilize this facility can reference the subschema entry

which defines the supplied name and can extract any of its characteristics (e.g., for items: type, PICTURE, PICTURE length, data length, and null-representation) as well as operate on the data values themselves.

This is the type of data independence that facilitates the implementation of general utilities, such as general text editors, dump routines, and on-line interactive query languages.

SPECIFIC EXTENSIONS

Item Type Extensions

There are many extensions at the item level. Extended data types include double-precision numbers, both integer and real, plus the types TIME, DATE, and CODE. TIME and DATE are self-explanatory, and in both cases a special internal format provides for compilation of these values on disk. CODE associates a literal string for external (human) consumption with space-saving internal code types. These codes are indices into a common stored decoding table. Coding in both directions is automatic.

The code table is treated as a single literal pool in each database. Literal strings which are represented by codes (whether the same or different) for more than one item are not repeated in separate per-item tables.

The Prime DBMS also permits variable-length character strings. It automatically stores such strings without any trailing blanks. The length attributed to these strings for accessing purposes is taken from the invoked subschema User Work Area at run-time. The reserved space need not be large enough to hold every value of the item in the database; only those which are accessed. The DBMS will fill with trailing blanks as necessary to reach the size required by the subschema.

No single value is used to represent a null value universally. The representation of the null value depends on the data type and is chosen for efficiency by Prime.

Data Aggregates

There is a very significant extension to data aggregates, and that is that any data aggregate may occur a variable number of times, regardless of whether it is part of, or contains, any other data aggregate. This feature cannot yet be utilized in a COBOL subschema, because it is beyond the ability of ANS COBOL.

One significant use of this feature works well with CODE item types to provide exception reporting without wasting space. The usual procedure for this is to encode the standard values (responses) with an additional code for exceptions (often, "Other, Specify").

The specification of this exception would be a string which rarely appeared. Defining it as a variable length and/or variable in occurrence would result in a minimum of waste in this situation.

FORTRAN Record Overlays

An extension concerning records is the ability to specify that different record types shall share the same space in a FORTRAN User Work Area (COMMON Blocks), in consonance with the FORTRAN concept of EQUIVALENCE.

OMISSIONS/RESTRICTIONS

No description of the DBMS would be complete without identifying current ommissions vis-a-vis the DBTG-71 Report. These are:

1. Set Mode is always system standard: B-trees, linked-to-owner.

2. Temporary areas are not implemented.

3. The CHECK clause - CHECK IS PICTURE is not implemented.

   CHECK IS RANGE is limited to numeric values

   CHECK IS LIST is confined to character strings

4. The ORDER DML Command is not implemented.

5. New Groups in the Subschema

   No data aggregates or naming groups are permitted in the subschema unless they also occur in the schema upon which that subschema is based.

6. Dynamic Sets are not implemented.

7. ENCODING/DECODING is not implemented.

PENDING

Some current omissions are already planned but not available in the first release

1. Procedures

   The procedure facility is complex and far-reaching. The state of this facility has delayed the availability of ON CLAUSES. Procedures will also be available in PRIVACY, CALC and CHECK clauses.

2. DBMS Access to MIDAS Files

Application programs can access both DBMS files and Multiple Index Data Access System (MIDAS) files. Presently each type of file must be accessed by using its own access system. Extensions to both MIDAS and DBMS will make it possible for MIDAS files to be accessed by DBMS commands.

3. Testing

Testing new DML programs is very time-consuming if a test database must be created. It is often difficult to obtain a meaningful test in a synthetic environment. Testing update DML programs on an active database can obviously have disastrous effects. The Prime DBMS facilites testing of update DML programs by providing test mode.

When a DML program invokes a schema for test mode, all transactions are considered retrieval transactions. Differential file techniques are used to store the updates in a scratch area that is private to the test mode program. The database is never modified in any way by a DML program in test modes. Other run-units, whether in test mode or not, are completely unaffected by a run-unit in test mode.

SECTION 6

DBMS FILES

INTRODUCTION

This section defines the database types of files that serve the Database Management System.

The database files are subordinate to segment directories, one segment directory per volume per schema. A database file may, thus, be uniquely identified by schema name, volume name, and segment directory entry number (INTEGER*4). This information is displayed by the VERIFY commands of the DBACP.

There are eight types of files associated with each schema or database:

- o Schema table (one per database:)
- o Subschema tables (one per subschema)
- o Area files (one per area)
- o Set files (one per set)
- o Calc files (one per record with location mode calc)
- o Log file (one per schema)
- o Before-Image file (one per schema)
- o After-Image file (one per schema)

SCHEMA TABLE

The schema table is a tabular representation of all of the information supplied in the schema source DDL. Also included in the schema table are: volume/entry identifier for each database file other than the schema table, keys for variable locks, and a directory of all subschemas for the schema.

SUBSCHEMA TABLE

The subschema table is a tabular representation of all of the information supplied in the subschema source DDL plus privacy information from the corresponding schema constructs.

AREA FILE

Area files are divided into equal length sections called buckets. Record occurrences are stored in the bucket from the top down. A directory of all record occurrences stored in a bucket grows upward from the bottom of the bucket. When there is no more room in the bucket for another directory entry and at least part of a record occurrence, the bucket is flagged as FULL, and the next bucket with available space is used.

SET FILE

Set files contain a Set Directory and B-trees (see Knuth Vol. 3, Section 6.2.4) representing set occurrences. The set directory entries are of fixed length for each set with all free entries linked together on an available entry list. There is one B-tree node space for each member list (one list per member if "ORDER IS SORTED;" else one per set) as well as one node space for each search key for all members of the set. The B-tree nodes are of fixed length for each node space with all free nodes linked together on an available node list for each node space.

CALC FILE

Calc files contain a hash table which is used to find the DBK of a record using a table entry number derived from hashing the concatenated values of all items declared as CALC keys for this record. At any time a table entry may be used or it may be flagged as empty (never used) or freed (used but record deleted or calc key(s) modified).

LOG FILE

The log file is a sequential file for writing messages regarding transactions and updates. The log file is intended to assist in recovery from serious database damage such as disk head crashes. The log file may grow very rapidly when there is a lot of transaction activity and update activity. The log file should be cleared periodically using the CLEAR LOG command of the DBACP when the database is inactive and secure. Message types are ASCII strings, start transaction, end transaction, abort transaction, and update a block. The pointer to the next available location in the log file is located in the transaction tables of the before-image file.

BEFORE-IMAGE FILE

The before-image file consists of the transaction tables and the before-images. The transaction tables contain transaction numbers, a list of completed transactions, information for the before-image queue, the log file, and the after-image file and other information necessary

for support of concurrent update of files. The rest of the before-image file contains before-images. Just before an update is made to a block in a database file, a copy of the block is saved in the before-image file. The blocks are arranged as a queue which is cyclically reused. Pointers describing the before-image queue are kept in the transaction tables.

AFTER-IMAGE FILE

The after-image file is a sequential file of after-images. An after-image is a copy of a modified block. After-images provide the redundancy required for a recovery after tragedies such as disk head crashes. An after-image is written when a block is depaged because of overflow of the buffer pool or at the end of a transaction. After-images are written at the end of the after-image file. A pointer to the next available location in the after-image file is located in the transaction tables of the before-image file.

SECTION 7

DATABASE GENERATION & ACCESS

INTRODUCTION

This section discusses what is involved in creating a database. It includes some important considerations of file allocation and the steps required to create a database.

SCHEMA CREATION

The Schema DDL Compiler generates the schema table and enters the new schema name and number and volume name for the schema table in the system directory of all schemas.

See the schema DDL manual for a complete description of the Data Definition Language and how to use the schema compiler.

FILE ALLOCATION

The files for a given database are created and initialized using the ALLOCATE FILES command of the DBACP (Section 9).

To assist the DBA in constructing the database, the DBACP queries him about quantities for which no fixed value was defined in the schema, such as:

- The average length of variable-length strings.
- The average number of occurrences of a variable-occurrence data aggregate.
- The maximum number of occurrences of each record type.
- The average number of manual inserts of each manual member of each set.

From the estimates given by the DBA, the DBMS calculates the exact storage requirements of each file constituting the database, and informs the DBA of space usage statistics including:

- The length and node-size of each B-tree.
- The number and size of each collection of buckets.
- The size of CALC file tables.
- The data-utilization ratio (the inverse of the overhead ratio).

The DBA need not proceed immediately using these figures; he can test various assumptions and adjust his estimates to arrive at a final configuration which he feels best suits the enterprise's current needs. In this way he may manage to reduce overhead, compare various versions for efficiency, or find a way to force the database into a smaller space occasioned by the size of his current system. Because of the ease of expansion, the DBA is free to select a version of the database which adjusts today's data needs to today's system, even though he knows this version to be much smaller than ultimately intended.

Having selected the satisfactory space configuration, the DBA can then actually allocate the files. The DBMS requests a volume identification for each AREA, SET, CALC TABLE, BEFORE-IMAGE FILE, the LOG, and the AFTER-IMAGE FILE associated with the schema. Thus, though DBMS depends on FMS, which cannot span volumes, a database can span volumes because each file is confined to a single volume itself, though the collection of files may cover all available volumes. The DBA assigns all files to available volumes, even those he originally defined only for future usage. AREAS defined in the schema which are not used initially can be defined to have one occurrence of a record in them during the space-calculation dialog, and thus require minimal space until they actually are put into use.

Only a DBA can allocate database files. A DBA can also clear the files of all data, or delete the files and data. These commands (ALLOCATE, CLEAR, and DELETE) must operate on all files of the database to ensure its integrity (refer to Section 9 for a description of these commands). The creation dialog is recorded, as are all DBACP dialogs, along with the date, time and name of the DBA. This record may be filed for later scrutiny to recall the origins of the database and the founding estimates.

After allocation of the database, it must be populated with data. The DBA defines a subschema for database loading and writes an application program in the language of his choice (COBOL or FORTRAN), which utilizes the DML to construct the database via updating DML commands. This gives the DBA complete control over the populating process.

The actual steps for creating a database are:

1. Define the schema and compile it into a schema table (Schema DDL Compiler).

2. ALLOCATE store space for the database (ALLOCATE files in DBACP).

3. Define a subschema to be used by the loading program and compile it into a subschema table (FORTRAN or COBOL Subschema DDL Compiler).

4. Write the database load program for this database.

5. Pre-process the load program into standard host-language form (FORTRAN or COBOL DML pre-processor).

6. Compile and run the loading program.

SECTION 8

DATABASE MAINTENANCE TECHNIQUES

INTRODUCTION

After the database is created and put into use, the DBA must monitor its use for loss of efficiency and provide for its expansion and re-organization when or if necessary. The need for reorganizing must be determined from space utilization figures. The PRIMOS system accounting data, which includes a measure of paging activity, can be used as an aid in this context.

A DBA can request space utilization status reports at any time. These reports detail the space utilization figures for all areas, sets, CALC tables and all recovery files for each requested database.

Whenever the DBA observes excessive paging activity, or the impending overflow of allocated space, or lengthening response times due to excessive accumulation of record alterations, he can initiate garbage collection.

MEDIA SPACE VERIFICATION

At any time in the life of a database, the status of the various files in terms of space usage may be ascertained using the various VERIFY commands of the DBACP. If file space usage is verified frequently, there is less chance of the DML command processor error "MEDIA SPACE NOT AVAILABLE" being encountered (see Media Space Clean-up and Expansion).

MEDIA SPACE CLEAN-UP AND EXPANSION

To make more media space available, space no longer in use (due to DELETE record DML commands) may be freed for DBMS use using the PACK AREA command on any area from which records have been deleted since the ALLOCATE FILES or the last PACK AREA command. If insufficient space is freed using the PACK AREA command, or media space is not available in one or more set or calc files, the DBACP command, EXPAND FILES, must be used. After an EXPAND has been executed, it is advisable to PACK any areas in which the bucket size was expanded to optimize space usage by trying to concatenate any record occurrences which were continued in other buckets.

The expansion dialog is similar to file allocations, expanding only those spaces which actually require expansion. Since the revised definition is at least partially based on actual usage, it could turn out to be more efficient. The DBA may wish to investigate expansion simply to see if a more efficient configuration results from using

actual figures.

If many records with location mode calc are deleted or the calc keys modified, the calc file may become filled with freed (no longer used) hash table entries, thus increasing the number of hash table probes for success or failure. This condition may be remedied by using the PACK CALC command of the DBACP which rehashes the entire table.

CONCURRENCY

A run-unit which updates a database must have exclusive use of the database files unless concurrent update is allowed (see ALLOW CONCURRENT UPDATE in DBACP). Since the concurrent update mechanism makes use of before-images, a before-image file of "sufficient" size must have been allocated and before-imaging must be turned on (see START BEFORE-IMAGING in DBACP) before concurrent update can be allowed.

The size of the before-image file is requested from the user by DBACP during the ALLOCATE FILES dialog as a percentage of the total database file size. This can be estimated by multiplying the percentage of the database accessed by an average update transaction times the maximum number of concurrent update run-units using this schema. The percentage of the database accessed is roughly the ratio of the number of record occurrences accessed by an average update transaction to the total number of record occurrences to be stored (i.e., the sum of the answers given for the individual record types in the ALLOCATE FILES dialog).

NOTE:

> Before-imaging may be either on or off when concurrent update is disallowed.

RECOVERY

There are two types of DBMS recovery available: roll back a database to a given point, or roll forward a database from a check point (see SAVE SCHEMA in DBACP) to a given point. Roll back is implemented using before-imaging and roll forward using after-imaging. The points to which a database is rolled back or forward is determined using the logging function of the DBMS (see the BEFORE-IMAGING, AFTER-IMAGING, and LOGGING commands in DBACP).

Although before-image, after-image, and log files are allocated and before-imaging, after-imaging, and logging may be turned on and off, the actual roll back and roll forward processors have not yet been implemented. These will be documented and distributed to DBMS users with Rev. 14 of PRIMOS. Until then, users are advised to backup active databases with the SAVE SCHEMA command in DBACP as described below.

NOTE:

Before-imaging does double duty with roll back and concurrent update.

DATABASE BACKUP

It is advisable to keep one or more copies of an active (frequently altered) database on magnetic tape for backup using the SAVE SCHEMA command of the DBACP. It is also advisable to save a database before using any volatile DBACP commands such as PACK AREA or running a DML program which "cleans up" the database by deleting record occurrences (periodic purging).

DML COMMAND PROCESSOR CLEAN-UP

If a DML program does not terminate properly, it can leave the data base in an indeterminate state. Termination can be caused by the program user hitting the break key, by the DML program exiting before ending a transaction, by fatal errors such as division by zero or by a system halt. After a DML program halts abnormally, the DML clean- up program, CLUP, should be run from the same terminal. CLUP rolls back the current transaction if one is active, closes files, releases locks, and cleans up DBMS control tables. If a DML user is uncertain about the termination of the DML Program, the user should run CLUP. If the user is not found in the DBMS, CLUP has no effect. Executing CLUP can never have an adverse effect.

PRIVACY KEYS

Privacy keys for variable locks may be displayed or altered at any time (see various KEY and KEYS commands in DBACP). This does not affect any run-unit which has already gained access to the data whose keys are concurrently changed.

SECTION 9

DBA COMMAND PROCESSOR (DBACP)

INTRODUCTION

The DBACP (Database Administrator Command Processor) is an interactive processor which allows the Data Administrator to create, investigate, and change the various files within databases. Several commands are available to all users for verification of the current status of database files. All other commands are restricted to use by those people officially designated as Data Administrators.

PROCEDURES

To initiate DBACP, the user must log in and type:

DBACP

DBACP prompts for each new command by displaying:

READY

SCHEMA schema-name READY

The user may exit DBACP by typing the command:

QUIT or Q

After each command the elapsed, CPU, and disk I/O timings are displayed in seconds.

Concurrent Run-Units

Those DBACP commands which alter an existing database or set system flags which limit run-unit access to the schema may only be executed when no run-units are currently running against the schema specified. Those commands which affect database files, such as the EXPAND and PACK commands, automatically lock the schema as well as checking for concurrent usage to prevent further invocation of the schema while the command is being executed (the UNLOCK SCHEMA command must be used to free the schema). If no run-units are currently accessing the schema, the command proceeds. If there are run-units currently accessing the schema the run-unit user numbers are displayed and the user is asked:

DO YOU WISH TO ABORT THE RUN-UNITS?

If yes is typed, the run-units are aborted and the command proceeds, else the user is asked:

DO YOU WISH TO WAIT FOR THE RUN-UNITS TO FINISH?

If no is typed, the command is aborted; otherwise the command waits, giving a running list of run-unit user numbers accessing the schema as they are invoked and exit until all are finished. At this time the command is executed to completion.

SECURITY

Data Administrators

Data Administrators are identified by login name. There exists a protected system list of Data Administrator names, including a fixed number of privileged Data Administrators. A privileged Data Administrator need not specify privacy keys for schema privacy locks for ALTER, DISPLAY, and LOCKS.

Schema Privacy Locks

Privacy locks may be specified in the schema DDL for ALTER (ALLOCATE, CLEAR, DELETE, EXPAND, PACK, etc.), LOCKS (all commands with object KEY or KEYS), and DISPLAY (VERIFY or implicit verification). If locks exist for any of the above functions when execution is attempted, the user will be prompted for a privacy key for the related schema lock. If an invalid key is specified, the command is aborted. Once a valid key is specified, the function remains unlocked for the current schema until the next schema name (possibly the same) is specified in a command.

COMMAND SYNTAX

Data Administrator commands are of the general form:

verb object schema-name

The command verb must always be specified. If the command object is omitted, the object, SCHEMA, is assumed. When a command object specifies some part or function of a schema, the schema-name may be used to specify the desired schema. Once a schema has been specified by a schema-name on an appropriate command, the schema is the current schema until another schema-clause is specified. If the schema-name is omitted from a command, the current schema is assumed. The name of the current schema, if any, is displayed with each READY prompt for a new command.

The format of the schema-name is:

OF SCHEMA schema-clause

NOTE:

The words OF and SCHEMA are not required.

DBACP COMMAND VERBS AND OBJECTS

The following are lists of valid command verbs and objects. Those command words or word parts which are not underlined may be omitted.

VERBS

ADD
ADVANCE
ALLOCATE
ALLOW CONCURRENT UPDATE
CHANGE
CLEAR
DELETE
DISALLOW CONCURRENT UPDATE
EXPAND
LOAD
LOCK
PACK
QUIT
RENAME
RESTORE
SAVE
START
STOP
UNLOAD
UNLOCK
VERIFY

OBJECTS

AFTER IMAGING
AREA area-name
AREAS
BEFORE IMAGING
CALC OF RECORD record-name
CALCS
FILES
KEY OF LOCK lock-name
KEYS
LISTING *
LOGGING
SCHEMA schema-name *
SCHEMAS *
SET set-name
SETS
SUBSCHEMA subschema-name
SUBSCHEMAS
TAPE *

* no schema-name with this object

COMMAND DESCRIPTION

The following is a description of each valid DBACP command grouped by command object and listed in alphabetical order.

****
* AFTER-IMAGING *
****

Function

After-imaging is the process of writing two copies of a database block. When updating the database, the first copy is put into the database file; the second is written to a collector file which contains after-images. After-imaging allows the database to be restored (roll forward) to its latest state in the event the files are destroyed.

Format

```
{CLEAR  }
{START  }
{STOP   } AFTER-IMAGING [OF SCHEMA schema-name]
{VERIFY }
```

Descriptions:

CLEAR

Delete all entries from after-image file and reinitialize.

START

Set a system flag so that all run-units using this schema invoked after this point will save after-images in the after-image file for use with roll forward facilities.

STOP

Set a system flag so that all run-units using this schema invoked after this point will not save after-images.

VERIFY

Display the volume name and entry number for the after-image file. Display a message stating if after-imaging is on or off.

****
* AREA *
****

Function

The object 'AREA' allows the Database Administrator to perform maintenance functions on an area such as restoration of files, file cleanup and packing, copying a file onto a magnetic tape or displaying information about the area file.

Format

```
{LOAD  }
{PACK  }
{UNLOAD} AREA area-name [OF SCHEMA schema-name]
{VERIFY}
```

Descriptions:

LOAD

Delete the dummy area file and restore the original area file from magnetic tape.

PACK

Erase all record occurrences in the area file that have been flagged as DELETED. Squeeze the occurrences and bucket directories as records are erased. After all deleted records are erased and their occurrences and directory entries are freed, all broken record occurrences (in more than one bucket) are examined and, if possible, concatenated in the newly freed spaces. The number of records erased and concatenated is displayed.

NOTE:

The before-image, after-image, and log files are automatically cleared by the PACK command since the images and log entries are no longer valid.

WARNING:

The "PACK AREA" command actually moves data in the area file, and will leave the file in an undefined state if aborted by user or operator intervention. It is therefore recommended that the schema be saved before one or more "PACK AREA" commands are executed.

UNLOAD

Copy the area file onto magnetic tape and replace the area file with a

dummy file containing information identifying the tape.

VERIFY

Display the following information about the area file: area name, number of buckets, and bucket size in bytes; volume name and entry number; login name, date, and time of creation; and percent empty, number of full buckets, and number of bytes free. If the area file has been unloaded, the date and time of unload are displayed instead of the usage statistics.

****
* AREAS *
****

Function

Area is a logical subdivision of a database. In Prime DBMS, Area also corresponds to a physical file on a disk. The object 'AREAS' allows the Database Administrator to list each area name and usage statistics.

Format

VERIFY AREAS [OF SCHEMA schema-name]

Descriptions:

VERIFY

Perform "VERIFY AREA area-name" for every area of the schema.

****
* BEFORE-IMAGING *
****

Function

Before-imaging is the process of making a copy of a database block before the block is updated. This allows the changes to a database to be cancelled (ROLL BACK) if an update to the item was found to be in error.

NOTE:

The Prime DBMS uses Before-Imaging Concurrent update protection.

Format

```
{CLEAR  }
{START  }
{STOP   } BEFORE IMAGING [OF SCHEMA schema-name]
{VERIFY }
```

Descriptions:

CLEAR

Reinitialize all concurrent update and recovery information for the schema, including the log, before-image, and after-image files.

START

Set a system flag so that all run-units using this schema invoked after this point will save before-images in the before-image file for use with concurrent update and roll back facilities.

STOP

Set a system flag so that all run-units using this schema invoked after this point will not save before-images: Before-imaging may not be stopped if concurrent update is currently allowed.

VERIFY

Display the size in bytes, volume name, and entry number of the before-image file. Display a message stating if before-imaging is on or off.

****
* CALC OF RECORD *
****

Function

CALC is a method of locating a record; where fields in the record are used as input to calculate the address of the record. CALC is one of three strategies used to locate records. The other two are direct and SET.

Format

{LOAD / PACK / UNLOAD / VERIFY} CALC OF RECORD record-name [OF SCHEMA schema-name]

Descriptions:

LOAD

Delete the dummy calc file and restore the original calc file from magnetic tape.

PACK

Rehash the CALC table, eliminating freed (deleted) entries, thus reducing the number of probes to failure or success.

NOTE: The before-image, after-image, and log files are automatically cleared by the PACK command since the images and log entries are no longer valid.

NOTE: The "PACK CALC" command generates a second file for the packed CALC table and deletes the old file after rehashing into the new one. There must be, therefore, enough space on the CALC file's volume for a temporary, duplicate file.

UNLOAD

Copy the CALC file onto magnetic tape and replace the CALC file within a dummy file containing information identifying the tape.

VERIFY

Display the following information about the CALC file: record name and number of entries in the table; volume name and entry number; login name, date, and time of creation; and number of table entries empty, used, and freed. If the CALC file has been unloaded, the date and time of unload are displayed instead of the usage statistics.

****
* CALCS *
****

Function

The CALCS object of the verb 'VERIFY' locates every record with location mode CALC and displays each record name with various usage statistics.

Format

VERIFY CALCS [OF SCHEMA schema-name]

Descriptions:

VERIFY

Perform "VERIFY CALC OF RECORD record-name" for every record of the schema with location mode calc.

****
* FILES *
****

Function

FILES are the physical mapping of a database to the storage medium (DISK).

The object 'FILES' allows all files within the named schema to be allocated, cleared, expanded, displayed or deleted.

Format

{ALLOCATE | CLEAR | DELETE | EXPAND | VERIFY} FILES [OF SCHEMA schema-name]

Descriptions:

ALLOCATE

Estimate, allocate and initialize all area, set, and CALC files and the log, before-image, and after-image files for this schema. Estimation is an interactive process, requesting size information about various schema constructs, and displaying the estimated file sizes and characteristics. Allocation and initialization occur only after the user accepts the estimation given. Size information requested includes the following: maximum number of occurrences of each record type to be stored within each area; maximum number of occurrences of each record type to be inserted into all occurrences of sets in which the records participate as manual members; mean value of all items which are used in variable occurs clauses; and the mean number of characters of items of type variable length string. The allowable range of each of these requested figures is displayed if either an invalid answer or a null line (carriage return only) is entered. Using these figures, the following are calculated and displayed: for each area, the number of buckets and bucket size in bytes; for each non-singular set, the number of 1Ø byte directory entries (one per owner); for each member and search list type of a set, the member or search list number, number of nodes and node size in bytes; for each record with location mode CALC, the number of 1Ø byte entries in the CALC table; for each record type, the number of data bytes and system bytes (counts and pointers) for an "average" record occurrence; the total space used by all area, set, and CALC files in bytes; and the ratio of data to total space for the maximum size database as a percentage. At this point the user may continue with allocation or abort the command. Information requested for file allocation includes a user estimate of the size of the before-image file as a percentage of total file space, and the name of each volume on which the files should be allocated. A short verification (see VERIFY FILES) is displayed for each file after it is

initialized.

CLEAR

Verify and reinitialize all area, set, and CALC files and the log, before-image, and after-image files. Verification includes a full verify of each file (see VERIFY AREA, VERIFY SET, etc.), after which the user may continue with or abort the CLEAR command. Verification may be circumvented altogether ("DO YOU WISH TO VERIFY?"). A short verification (see VERIFY FILES) is displayed for each file before it is reinitialized.

DELETE

Verify and delete all area, set, and CALC files and the log, before-image, and after-image files. Verification proceeds as in the CLEAR FILES command.

EXPAND

Verify, examine, reestimate, and, if necessary, reallocate and expand all area, set, and CALC files. The log, before-image, and after-image files are reallocated and reinitialized. Verification proceeds as in the CLEAR FILES command. Reestimation is identical to the estimation process in the ALLOCATE FILES command with the following exceptions: reestimation takes in to account the current usage of each file by examining the amount of free space left, total amount of data stored, and various statistics on distribution of data within each file; instead of requesting "MAXIMUM NUMBER OF OCCURRENCES" of each record type within areas and inserted manually into sets, the request is for "MAXIMUM NUMBER OF ADDITIONAL OCCURRENCES", meaning additional occurrences to be stored and inserted from this point. Only those area, set, and CALC files which need more space are reallocated and expanded. Otherwise, reallocation and expansion proceed as allocation and initialization in the ALLOCATE FILES command, with the exception that the short verification is displayed for each file before it is expanded. NOTE: The "EXPAND" command generates a new file, initializes the new file, copies all data from the old file, and deletes the old file for each file to be expanded. There must be room, therefore, for a new file to exist before the corresponding old file is deleted if the new file is to be on the same volume as the old file.

VERIFY

Display a short verification of all area, set, and CALC files and the log, before-image, and after-image files. If the files have not been allocated or have been deleted, short verification (as well as long verification) includes only the schema construct name and the message "<NOT ALLOCATED>". Otherwise, short verification includes the following: schema construct name; and the volume name and entry number. Verify of areas includes number of buckets and bucket size in bytes. Verify of CALC files includes the number of table entries.

****
* KEY OF LOCK *
****

Function

Those privacy clauses in the schema DDL using lock-names have no effect until a literal key is assigned. The object KEY allows the Data Administrator to assign, deallocate, change, and display the literal values for the keys for variable locks.

Format

```
{ALLOCATE}
{CHANGE  }
{DELETE  } KEY OF LOCK  lock-name  [OF SCHEMA schema-name]
{VERIFY  }
```

Descriptions:

ALLOCATE

Set the key for this lock if it is not set.

CHANGE

Verify and reset the key for this lock.

DELETE

Deallocate the key for this lock.

VERIFY

Display the key for this lock.

****
* KEYS *
****

Function

Like the object "KEY", KEYS allows the Data Administrator to assign, deallocate, change, and display the literal values for the keys for all variable locks for the schema.

Format

```
{ ALLOCATE }
{ CHANGE   } KEYS  [OF SCHEMA schema-name]
{ DELETE   }
{ VERIFY   }
```

Descriptions:

ALLOCATE

Set the keys for all variable locks in this schema that are not set.

CHANGE

Verify and reset the keys for all variable locks in this schema.

DELETE

Deallocate the keys for all variable locks in this schema.

VERIFY

Display the keys for all variable locks in this schema.

****
* LISTING *
****

Function

LISTING allows the DBACP to retain a recording of the session with the Database Administrator. The Database Administrator can save or clear the listing.

Format

{CLEAR} LISTING
{SAVE }

Descriptions:

CLEAR

Rewind and truncate the DBACP listing file DBAØnn, where nn is the user number. Display and write out a heading including login name, terminal number, date, and time. When DBACP is invoked, a clear list is automatically performed.

SAVE

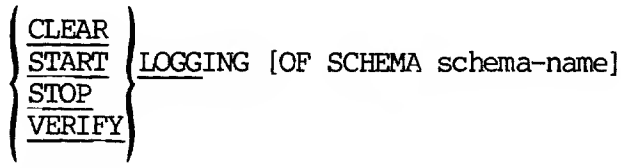
Copy the DBACP listing file into a user specified file. The DBACP listing includes a copy of all interactions between the user and DBACP except the values of keys of variable locks.

****
* LOGGING *
****

Function

The DBMS records all database transactions in a file called 'log file'. The object 'LOGGING' and the four verbs allows the Database Administrator to control the operation of this log file.

Format



Descriptions:

CLEAR

Delete all entries from log file and reinitiate.

START

Set a system flag so that all run-units using this schema invoked after this point will save all logging information in the log file.

STOP

Set a system flag so that all run-units using this schema invoked after this point will not save logging information.

VERIFY

Display the volume name and entry number for the after-image file. Display a message stating if after-imaging is on or off.

****
* SCHEMA *
****

Function

The object 'SCHEMA' allows the Database Administrator to perform maintenance functions on the named schema such as renaming, adding a new schema, deleting a schema, updating, etc.

Format

```
{ {ADD    }                                                    }
{ {RESTORE}                                                    }
{ ALLOW CONCURRENT UPDATE                                      }
{ DELETE                                                       }
{ DISALLOW CONCURRENT UPDATE                                   }
{ LOCK                                                         }
{ RENAME                                  [SCHEMA schema-name]*}
{ SAVE                                                         }
{ UNLOCK                                                       }
{ VERIFY                                                       }
```

* If the Schema-name is omitted, the current schema is assumed.

Descriptions:

ADD

Enter a schema which was created on another PRIME DBMS system into the Schema Directory for this system. The schema is assumed to have been transported intact from its original system on volumes on removable disk packs which have been loaded on this system. DBACP asks for the name of the volume on which the schema table resides and the schema number from the original system. If there are any conflicts with the schema name or number, the user may change the name or number of the schema being added or abort the command.

ALLOW CONCURRENT UPDATE

Set a system flag which will allow concurrent updating and retrieving of all files for this schema for all run-units currently running and invoked after this point. Before-imaging must be on before concurrent update can be allowed.

DELETE

Verify and delete all subschemas, all area, set, and calc files, the log, before-image, and after-image files, and the schema table for this schema. Verification proceeds as in the CLEAR FILES command with the addition of the schema table and the subschemas. When deletion is complete, the entry for the deleted schema is removed from the Schema Directory.

DISALLOW CONCURRENT UPDATE

Set a system flag which will allow only one updating or multiple retrieving run-units to access the database concurrently.

LOCK

Set a system flag which prevents any run-units invoked after this point from accessing this schema.

RENAME

Change the name of this schema. If desired, the schema number may also be changed (system supplied).

RESTORE

Copy the schema and all of its related files from magnetic tape and enter it into the Schema Directory. If there are any schema name or number conflicts, the command is aborted.

SAVE

Copy the schema and all of its related files onto magnetic tape.

UNLOCK

Set a system flag which allows run-units to access this schema after this point.

VERIFY

Display the following information about the schema table: schema name and number; volume name and entry number; login name, date, and time of creation; name of schema DDL source file; a message stating if concurrent update is allowed or not allowed; and messages stating if logging, before-imaging, and after-imaging are on or off.

****
* SCHEMAS *
****

Function

The object 'SCHEMAS' allows the display of the names and numbers of all SCHEMAS known to the DBMS.

Format

VERIFY SCHEMAS

Descriptions:

VERIFY

Display the schema name and number of each schema in the system.

****
* SET *
****

Function

The object 'SET' allows the Database Administrator to perform maintenance functions on the SET file.

Format

```
{LOAD  }
{UNLOAD} SET set-name [OF SCHEMA schema-name]
{VERIFY}
```

Descriptions:

LOAD

Delete the dummy set file and restore the original set file from magnetic tape.

UNLOAD

Copy the set file onto magnetic tape and replace the set file with a dummy file containing information identifying the tape.

VERIFY

Display the following information about the set file: set name; volume name and entry number; login name, date, and time of creation; number of directory entries and number of entries free (for non-singular sets only); and for each member and search list, the member or search list number, number of nodes, node size in bytes, and number of nodes free.

****
* SETS *
****

Function

The object 'SETS' allows each set name to be listed at the terminal with various usage statistics.

Format

VERIFY SETS [OF SCHEMA schema-name]

Descriptions:

VERIFY

Perform "VERIFY SET set-name" for every set of the schema.

****
* SUBSCHEMA *
****

Function

The 'SUBSCHEMA' object allows the Database Administrator to verify and delete the named subschema.

Format

```
{DELETE}
{VERIFY}  SUBSCHEMA  subschema-name  *[OF SCHEMA schema-name]
```

Descriptions:

DELETE

Verify and delete the subschema.

VERIFY

Display the following information about the subschema table: subschema language type, name, and number; volume and entry number; login name, date, and time of creation; name of subschema DDL source file; and size of common work space in bytes for DML programs accessing this subschema.

****
* SUBSCHEMAS *
****

Function

This object performs the same functions as the 'SUBSCHEMA' object on all subschemas associated with the named schema.

Format

```
{DELETE}
{VERIFY}  SUBSCHEMAS  [OF SCHEMA schema-name]
```

Descriptions:

DELETE

Verify and delete all subschemas of this schema. The user may abort the command after the verification of any subschema.

VERIFY

Perform "VERIFY SUBSCHEMA subschema-name" for every subschema of this schema.

****
* TAPE *
****

Function

The object TAPE allows the Database Administrator to control the magnetic tape such as rewind, advance tape forward, or to display information about the tape.

Format

$$\left\{\begin{array}{l}\underline{\text{ADVANCE}}\\ \underline{\text{CHANGE}}\\ \underline{\text{VERIFY}}\end{array}\right\}\ \underline{\text{TAPE}}$$

NOTE: The first command executed which accesses magnetic tape will request the tape unit number for that tape. All successive commands will assume the same tape unit until the physical end of tape marker is reached or a 'CHANGE TAPE' command is executed.

Descriptions:

ADVANCE

Advance the tape forward to the next schema save or file unload and verify the header.

CHANGE

Rewind the tape and request a new tape unit number (may be the same).

VERIFY

Read the tape header and display the following information: schema name and number; if file unload, name of schema construct; date and time saved or unloaded; and tape sequence number (always one for first tape and incremented by one for each successive tape for this schema save or file unload). If the tape is positioned at the end of the last save or unload on the tape, the message "END OF TAPE" is displayed.

APPENDIX A

PERFORMANCE NOTES

The following presents a brief summary of performance-oriented considerations:

1. The DBMS is designed for on-line processing and recovery. This reflects in the generation approach to database update, and the delayed garbage collection of records. It is part of the rationale for using B-trees as the sole implementation of sets.

2. Database Transactions (DBT's) provide for minimal data lockout and multiple concurrency (viz: retrieval and update).

3. Two levels of buffer - pooling create efficient physical I/O management and allow common information to be shared. The large CPU page size and its equivalence to physical block size are a source of I/O speed.

4. The databases are expandable, partly due to the plug-in expandability of Prime hardware, partly due to the ability of adjust for hardware expansion, and consumated in the EXPAND and PACK capabilities provided to the DBA by the Prime DBMS.

5. The ability of the DBA to examine every aspect of the schema, and of the DBMS to utilize actual running figures is realized in Prime's DBMS. DBA estimates during later expansion are an aid to increasing overall efficiency.

6. Rapid search is provided by several facilities: the hashing implementation of CALC, the use of B-trees, and the Set Owner-Directory which associates owners with all their members, which is accessed directly from every associated record occurrence (i.e., universal link to owner).

7. Space utilization is generally efficient: the size of DBMS-structured files compares favorably with pre-DBMS versions.

8. The hardware cache memory and random-sectoring disk controller technique enhance speed; the self-correcting internal memory and disk controller enhance integrity; the hardware ring structure creates additional security.

APPENDIX B

MAXIMUMS

The following numbers are definitions of maximum sizes with respect to the DBMS. The reader should be aware that many of these numbers assume no storage limitation but is available for future system expansion.

PRIME Storage capability	2.4 Billion Bytes
Maximum file size:	300 Million Bytes
Maximum Bucket size:	128,000 Bytes
Maximum Record Size:	65,000 Bytes
Maximum Number of Areas:	1023 Areas
Maximum Number of Record types:	1023 record types
Maximum Number of Buckets per Area:	(2**20)-1 Buckets
Maximum Number of Record Entries per Bucket:	255 entries
Maximum Number of Items in a Record:	4096 items
Maximum Number of users:	63 users
Maximum Number of concurrent files opened at the same time:	256 files
Maximum Number of Sets:	1023 Sets
Maximum Number of Set Occurrences:	(2**27)

INDEX

(DBACP) 4-7

(DBT'S) 5-1

(DBT'S) A-1

(DBTS) 4-4

(DMLCP) 4-6

(ROLL BACK) 9-10

ABORT TRANSACTION 4-5

ABORT TRANSACTION 4-4

ACCESS STRATEGIES 4-1

ADD 9-19

ADVANCE THE TAPE FORWARD 9-26

AFTER-IMAGE FILE 6-2

AFTER-IMAGE FILE 6-3

AFTER-IMAGE FILE 7-2

AFTER-IMAGE 8-2

AFTER-IMAGING 9-6

ALLOCATE DATABASE FILES 7-2

ALLOCATE 4-6

ALLOCATE 7-2

ALLOCATE 9-13

ALLOCATE 9-15

ALLOCATE 9-16

ALLOW CONCURRENT UPDATE 9-19

APPLICATION PROGRAMS 5-4

AREA DEFINITION 3-1

AREA FILE 6-2

AREA NAME 9-8

AREA 7-2

AREA 9-7

AREA, SET, AND CALC 9-14

AREAS CAN BE EXPANDED 3-6

AREAS 8-1

AREAS 9-9

B-TREE 3-4

B-TREE 7-1

B-TREES 5-3

B-TREES 6-2

BEFORE-IMAGE FILE 6-2

BEFORE-IMAGE FILE 7-2

BEFORE-IMAGE QUEUE 6-3

BEFORE-IMAGES 4-5

BEFORE-IMAGING 8-2

BEFORE-IMAGING 9-10

BUCKET SIZE 9-8

BUCKETS 3-1

BUCKETS 3-6

BUCKETS 6-2

BUCKETS 7-1

BUFFER POOL 2-5

BUFFER POOL 6-3

BUFFER POOLING 2-4

CALC FILE TABLES 7-1

CALC FILE 6-2

CALC FILE 9-11

CALC FILES 9-14

CALC FUNCTION 4-1

CALC KEYS 6-2

CALC KEYS 8-1

CALC OF RECORD 9-11

CALC TABLE 7-2

CALC TABLE 9-11

CALC TABLES 8-1

CALCS 9-12

CARETAKER FUNCTIONS 4-8

CHANGE 9-15

CHANGE 9-16

CHANGES TO SCHEMA 4-2

CHANGES TO SUBSCHEMAS 4-2

CHECK CLAUSE 5-3

CHECK IS LIST 5-3

CHECK IS PICTURE 5-3

CHECK IS RANGE 5-3

CLEAR 7-2

CLEAR 9-14

CLEAR 9-17

CLEAR 9-18

CLEAR 9-6

CLUP 8-3

COBOL SUBSCHEMA 5-2

COBOL 4-2

CODE TABLE 5-2

CODE TYPES 5-2

COMMAND DESCRIPTION 9-5

COMMON BLOCK 4-2

COMMON BLOCKS 5-3

COMMON WORK SPACE 9-24

CONCURRENCY CONFLICT 4-5

CONCURRENCY 8-2

CONCURRENT ACCESS 4-3

CONCURRENT RUN UNITS 9-1

CONCURRENT UPDATE 9-10

CONCURRENT UPDATE 4-7

CONCURRENT USE 2-5

COPY THE SET FILE 9-22

COPYING 9-7

CREATING THE DATABASE 4-9

CYCLIC 4-1

DATA AGGREGATES 5-3

DATA AGGREGATES 5-2

DATA INDEPENDENCE 4-2

DATA INDEPENDENCE 5-1

DATA LENGTH 5-2

DATA MANIPULATION LANGUAGE 4-2

DATA STRUCTURE ASSISTANCE 2-5

DATA STRUCTURES 4-1

DATA-UTILIZATION RATIO 7-1

DATABASE ADMINISTRATOR'S COMMAND PROCESSOR (DBACP) 4-7

DATABASE ADMINISTRATOR SUPPORT 4-8

DATABASE BACKUP 8-3

DATABASE FILE 6-1

DATABASE KEYS 4-1

DATABASE LAYOUT 3-1

DATABASE MANAGEMENT SYSTEM 6-1

DATABASE MODEL 2-5

DATABASE TRANSACTIONS (DBT'S) A-1

DATABASE TRANSACTIONS (DBT'S) 5-1

DATABASE TRANSACTIONS (DBTS) 4-4

DBACP COMMAND 8-1

DBACP DIALOGS 7-2

DBACP LISTING FILE 9-17

DBMS ARCHITECTURE 2-5

DBMS CONTROL TABLES 8-3

DBMS FILES 6-1

DBMS RECOVERY 8-2

DBMS STORAGE STRUCTURE 3-1

DDL 4-2

DEADLOCK 4-4

DEALLOCATE 9-15

DEFINITION TABLES 3-1

DELETE THE DUMMY SET FILE 9-22

DELETE 7-2

DELETE 9-15

DELETE 9-16

DELETE 9-19

DISALLOW CONCURRENT UPDATE 9-20

DISPLAY THE FOLLOWING INFORMATION ABOUT THE SET FILE 9-22

DISPLAY 9-8

DML COMMAND PROCESS (DMLCP) 4-6

DML COMMAND PROCESSOR CLEAN-UP 8-3

DML 4-2

DMLCP 4-7

DUMMY CALC FILE 9-11

DYNAMIC REFERENCE 5-1

DYNAMIC REFERENCE 4-2

DYNAMIC SETS 5-3

EASE OF EXPANSION 3-6

ENCODING/DECODING 5-3

END OF A TRANSACTION 6-3

END OF TAPE 9-26

END TRANSACTION COMMAND 4-4

END/ABORT 5-1

ENDING A TRANSACTION 8-3

ENTER A SCHEMA 9-19

ERROR-CORRECTING ARRAY 2-2

EXIT/ABORT 5-1

EXPAND FILES 8-1

EXPAND 4-6

EXPAND 8-1

EXPAND 9-1

EXPAND 9-14

EXPANSION 3-6

EXTENSIONS TO THE DBTG-71 REPORT 5-1

EXTENSIONS/RESTRICTIONS 5-1

FANOUT 3-6

FATAL ERRORS 8-3

FDR 1-3

FIELDS IN THE RECORD 9-11

FILE ALLOCATION 7-1

FILE AND ACCESS MANAGEMENT 2-4

FILE CLEANUP 9-7

FILE MANAGEMENT SYSTEM 2-4

FILES 9-13

FINAL DOCUMENTATION RELEASE 1-3

FMS 2-4

FORTRAN DDL 4-2

FORTRAN RECORD OVERLAYS 5-3

FORTRAN USER WORK AREA (COMMON BLOCKS) 5-3

FORTRAN 4-2

GENERIC UTILITIES 4-3

HARDWARE UPGRADE AND EXPANSION 2-3

HARDWARE 2-1

HASH TABLE 6-2

HASHING VALUES 4-1

HIERARCHICAL 4-1

HOST-LANGUAGE 7-3

IDR 1-3

IMPLEMENTATION OF DBTS 4-5

INITIAL DOCUMENTATION RELEASE 1-3

INTEGRITY AND SECURITY 2-2

INTEGRITY 4-3

INVOKE COMMAND 5-1

INVOKE 5-1

ITEM TYPE EXTENSIONS 5-2

KEY OF LOCK 9-15

KEYS OF VARIABLE LOCKS 9-17

KEYS 5-1

KEYS 9-16

LINKED-TO 5-3

LISTING 9-17

LITERAL POOL 5-2

LOAD 9-11

LOAD 9-22

LOAD 9-7

LOCATING A RECORD 9-11

LOCATION MODE 3-6

LOCATION MODE 8-1

LOG FILE 4-8

LOG FILE 6-2

LOG 7-2

LOGGING 9-18

LOGIN NAME 9-11

LOST UPDATES 4-4

MAGSAV/MAGRST 4-8

MAINTAINABILITY 2-5

MANUAL INSERTS 7-1

MAXIMUM BUCKET SIZE B-1

MAXIMUM FILE SIZE B-1

MAXIMUM NUMBER OF AREAS B-1

MAXIMUM NUMBER OF BUCKETS B-1

MAXIMUM NUMBER OF CONCURRENT FILES OPENED AT THE SAME TIME B-1

MAXIMUM NUMBER OF ITEMS IN A RECORD B-1

MAXIMUM NUMBER OF OCCURRENCES OF EACH RECORD TYPE 7-1

MAXIMUM NUMBER OF RECORD TYPES B-1

MAXIMUM NUMBER OF RECORD ENTRIES PER BUCKET B-1

MAXIMUM NUMBER OF SETS B-1

MAXIMUM NUMBER OF SET OCCURRENCES B-1

MAXIMUM NUMBER OF USERS B-1

MAXIMUM RECORD SIZE B-1

MAXIMUMS B-1

MEDIA SPACE VERIFICATION 8-1

MESSAGE TYPES 6-2

MIDAS 5-4

MULTIPLE DATABASES 3-1

NAMING GROUPS 5-3

NETWORK 4-1

NEW GROUPS 5-3

NODE-SIZE 7-1

NODES OF B-TREES 3-4

NUMBER OF BUCKETS 9-8

NUMBER OF BYTES FREE 9-8

NUMBER OF FULL BUCKETS 9-8

OBJECT TABLE 4-2

OCCURRENCES OF A VARIABLE-OCCURRENCE DATA AGGREGATE 7-1

OMISSIONS/RESTRICTIONS 5-3

ON ERROR CLAUSE 5-1

ON-LINE PROCESSING AND RECOVERY A-1

ORDER DML COMMAND 5-3

ORDER IS SORTED 6-2

ORIGINAL CALC FILE 9-11

OVERFLOW OF ALLOCATED SPACE 8-1

OWNER OCCURRENCE 3-4

PACK AREA 8-3

PACK CALC 9-11

PACK 3-6

PACK 4-6

PACK 9-1

PACK 9-11

PACK 9-7

PACKING 9-7

PAGE PROXIMITY 3-6

PAGE SIZE 2-1

PAGING ACTIVITY 8-1

PAGING SPEED 2-1

PASSWORD CAPABILITIES 4-3

PASSWORD PROTECTION 2-4

PDR 1-3

PERFORMANCE SUMMARY A-1

PHYSICAL INTEGRITY 4-6

PICTURE LENGTH 5-2

PICTURE 5-2

POINTER ARRAYS 3-4

POINTERS 6-3

PRELIMINARY DOCUMENTATION RELEASE 1-3

PRESERVING THE PHYSICAL INTEGRITY OF THE DTABASE 4-6

PRIME STORAGE CAPABILITY B-1

PRIMOS ARCHITECTURE 2-3

PRIVACY KEYS FOR VARIABLE LOCKS 8-3

PRIVACY KEYS 5-1

PRIVACY KEYS 8-3

PRIVACY 4-3

PRIVACY 5-1

PRIVACY-LOCKING 4-8

PROCEDURE FACILITY 5-4

PROTECT CAPABILITIES 4-3

RAM 2-4

RANDOM ACCESS MANAGER 2-4

RANDOM-SECTORING TECHNIQUE 3-6

RE-ENTRANT 2-5

READ THE TAPE HEADER AND DISPLAY 9-26

RECORD DEFINITIONS 3-1

RECORD NAME AND NUMBER OF ENTRIES IN THE TABLE 9-11

RECORD OCCURRENCES 8-3

RECORD OCCURRENCES 6-2

RECORD TYPEE 4-1

RECORD-TYPES 3-6

RECOVERY FACILITIES 4-7

RECOVERY PROCESS 4-8

RECOVERY PROCESSOR 4-7

RECOVERY 4-6

RECOVERY 8-2

RECOVERY 9-10

REFERENCE DOCUMENTS 1-5

RENAMING 9-19

RESTORATION OF FILES 9-7

RESTORE 9-11

RETRIEVAL DBTS 4-4

RETRIEVING OF ALL FILES 9-19

REWIND THE TAPE 9-26

RING STRUCTURE 2-2

ROLL BACK A DATABASE 8-2

ROLL BACK FACILITIES 9-10

ROLL BACK 4-7

ROLL BACK 8-2

ROLL FORWARD 4-8

ROLLBACK A TRANSACTION 4-5

ROLLBACK OF UPDATES 4-4

RUN-TIME 4-2

RUN-TIME 5-2

RUN-UNIT 4-6

RUN-UNIT 4-7

RUN-UNIT 5-1

RUN-UNIT 8-2

SAVE 9-17

SAVE/RESTORE 4-6

SAVE/RESTORE 4-8

SCHEMA CHANGES 4-2

SCHEMA CREATION 7-1

SCHEMA DDL COMPILER 7-1

SCHEMA DIRECTORY 3-1

SCHEMA PRIVACY LOCKS 9-2

SCHEMA TABLE 6-1

SCHEMA TABLE 7-1

SCHEMA 9-19

SCHEMAS 9-21

SEARCH KEY 6-2

SEARCH KEYS 2-6

SECURITY WITHIN PRIMOS 2-4

SECURITY 9-2

SEGMENT DIRECTORIES 6-1

SEQUENTIAL FILE 6-2

SET AND RECORD DEFINITIONS 3-1

SET DEFINITIONS 3-1

SET DIRECTORY 6-2

SET FILE 6-2

SET MODE 5-3

SET OCCURRENCES 6-2

SET ORDERING 2-6

SET SET-NAME 9-22

SET 7-2

SETS 8-1

SETS 9-23

SHARE COMMON PROGRAM SPACE 2-5

SHARED DATA 4-3

SHARED SINGLE-COPY CODE 2-5

SHARING OF COMMON FILES 2-5

SHARING/CONCURRENT ACCESS WITH MOST DBMS 4-3

SHARING/CONCURRENT ACCESS WITH PRIME'S DBMS 4-4

SPACE UTILIZATION 3-7

SPECIFIC EXTENSIONS 5-2

SPEED 2-1

SPEED 2-5

START TRANSACTION DML COMMAND 4-4

START 5-1

START 9-10

START 9-18

START 9-6

STATUS REPORTS 8-1

STOP 9-10

STOP 9-18

STOP 9-6

STORAGE SPACE 3-6

SUBROUTINE CALLS 2-1

SUBSCHEMA DATA DEFINITION LANGUAGE 4-2

SUBSCHEMA DIRECTORY 3-1

SUBSCHEMA TABLE 6-1

SUBSCHEMA 9-24

SUBSCHEMAS 9-25

SYSTEM DIRECTORY 7-1

TABLES OF LOCKS 2-5

TEMPORARY AREAS 5-3

TERMINATION 8-3

TESTING NEW DML PROGRAMS 5-4

TESTING 5-4

THE OPERATION OF THE RUN-UNIT 4-6

TIME OF CREATION 9-11

TIME, DATE, AND CODE 5-2

TRANSACTION ROLLBACK 4-5

TRANSACTION TABLES 6-2

TREE 4-1

TYPE 5-2

TYPES OF FILES 6-1

UNLOAD 9-11

UNLOAD 9-7

UPDATE DBT 4-5

UPDATE DBTS 4-4

UPDATE TRANSACTION NUMBERS 4-5

USER WORK AREA 5-2

VARIABLE PRIVACY LOCKS 5-1

VARIABLE-LENGTH CHARACTER STRINGS 5-2

VARIABLE-LENGTH STRINGS 7-1

VERIFY AND DELETE ALL SUBSCHEMAS 9-19

VERIFY AND DELETE ALL SUBSCHEMAS 9-25

VERIFY AND DELETE THE NAMED SUBSCHEMA 9-24

VERIFY AREA AREA-NAME 9-9

VERIFY CALC OF RECORD 9-12

VERIFY COMMANDS 6-1

VERIFY SCHEMAS 9-21

VERIFY SET SET-NAME 9-23

VERIFY SETS 9-23

VERIFY 9-8

VERIFY 9-10

VERIFY 9-14

VERIFY 9-15

VERIFY 9-16

VERIFY 9-18

VERIFY 9-24

VERIFY 9-6

VOLUME AND ENTRY NUMBER 9-24

VOLUME NAME AND ENTRY NUMBER 9-8

VOLUME NAME AND ENTRY NUMBER 9-11

VOLUME/ENTRY IDENTIFIER 6-1

WORKING STORAGE 4-2